Bernd Müller

Herstellung der Funktionalität eines Einzelradtesters für Off-Road-Reifen (agrar, build, forrest, military)

DIPLOMARBEIT

HOCHSCHULE MITTWEIDA

UNIVERSITY OF APPLIED SCIENCES

Fakultät Elektro- und Informationstechnik

Mittweida, 2011

Bernd Müller

Herstellung der Funktionalität eines Einzelradtesters für Off-Road-Reifen (agrar, build, forrest, military)

eingereicht als

DIPLOMARBEIT

an der

HOCHSCHULE MITTWEIDA

UNIVERSITY OF APPLIED SCIENCES

Fakultät Elektro- und Informationstechnik

Mittweida, 2011

Erstprüfer: Prof. Dr.-Ing. Olaf Hagenbruch

Zweitprüfer: Dr. Hartmut Döll

Vorgelegte Arbeit wurde verteidigt am:

Bibliographische Beschreibung:

Müller, Bernd:

Herstellung der Funktionalität eines Einzelradtesters für Off-Road-Reifen (agrar, build, forrest, military) - 2011. - 107 S.
Mittweida, Hochschule Mittweida, Fakultät Elektrotechnik und Informationstechnik, Diplomarbeit, 2011

Referat:

Off-Road-Reifen, insbesondere landwirtschaftliche Reifen, werden mit extremen Parametern (hohe Radlasten bis 120 kN und niedrige Reifendrücke zwischen 0,4 bis 2,6 bar) gefahren. Demzufolge zeigen diese Reifen sehr große Deformationen. Entsprechend der Reifenkonstruktion sind Deformationseigenschaften unterschiedlich und entscheiden über Bodenbelastung und Traktion. Die Beurteilung der agrotechnischen Eigenschaften, insbesondere Bodenbelastung, Traktion, Verschleiß und Dämpfung, erfolgen nach der Prüfmethode der TU Dresden. Dazu sind die Druckverteilung in der Kontaktfläche (siehe Abbildung 1), der Rollwiderstand und die Deformation der Reifen (siehe Abbildung 3) zu messen.

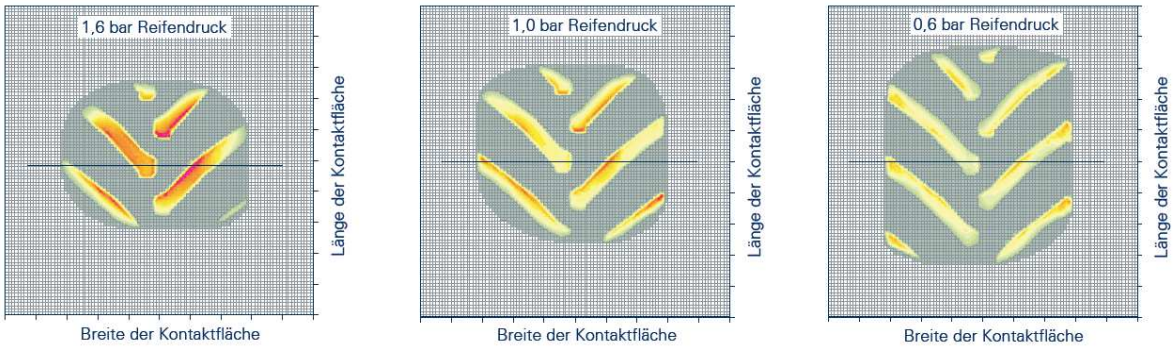


Abbildung 1: Druckverteilung in der Kontaktfläche [1]

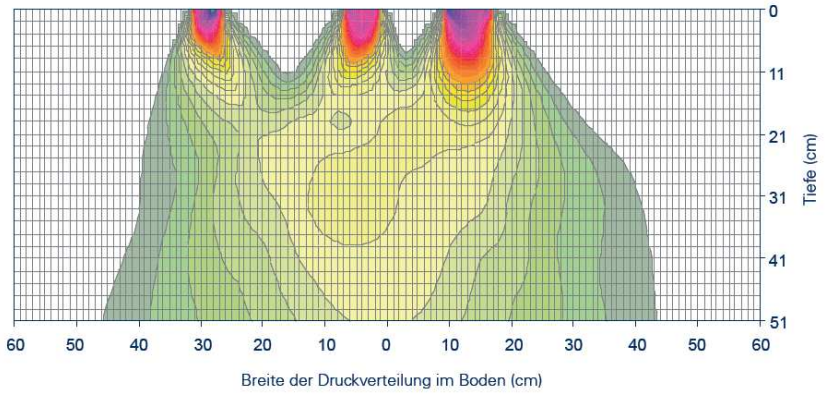


Abbildung 2: Druckverteilung im Boden (Druckzwiebel) [1]

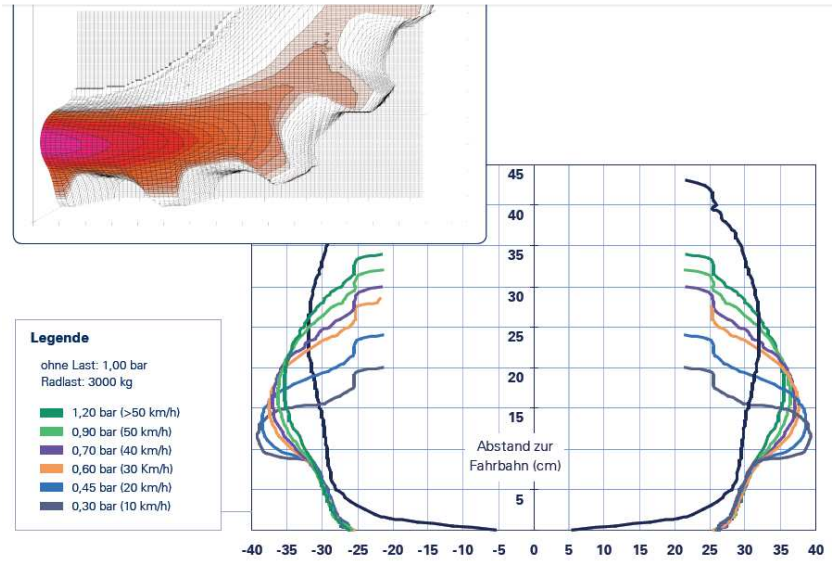


Abbildung 3: Veränderung der Reifenkontur [1]

Abbildung 4: Reifenprüfstand Gesamtansicht

Zur Arbeit am Reifenprüfstand waren entsprechend des Entwicklungstandes der Messtechnik, Messwerterfassung und Messwertaufbereitung (Standard-Hard- und -Software) von 1991 eine ständige Überwachung und manuelle Eingriffe in den Messvorgang erforderlich. Mit einer studentischen Arbeit ist 2010 der prinzipielle Nachweis zum automatischen Ablauf des gesamten Messvorganges untersucht worden. Bei der Umsetzung dieser Aufgabe haben sich im Praxisbetrieb Schwächen des Systems gezeigt, die abzustellen sind. Eine Erfassung und Verwertung von Daten ist dadurch nicht möglich.

Das Ziel der Diplomarbeit besteht darin, die Funktionalität des vorhandenen Reifenprüfstandes für den automatischen Betrieb, der Erfassung von Druckverteilung in der Kontaktfläche und des Rollwiderstandes herzustellen.

Nach einer gründlichen IST-Analyse der einzelnen Komponenten werden Lösungsvorschläge erarbeitet. Die Hardware-Elemente und die entwickelte Software wurden auf Praxistauglichkeit überprüft.

Zum Schluss erfolgen die Beschreibung und Auswertung des Gesamtentwurfs sowie Vorschläge für Erweiterungen und Anpassungen weiterer Prüfelemente (Querkraft-, Drehmomenteinfluss, Deformation).

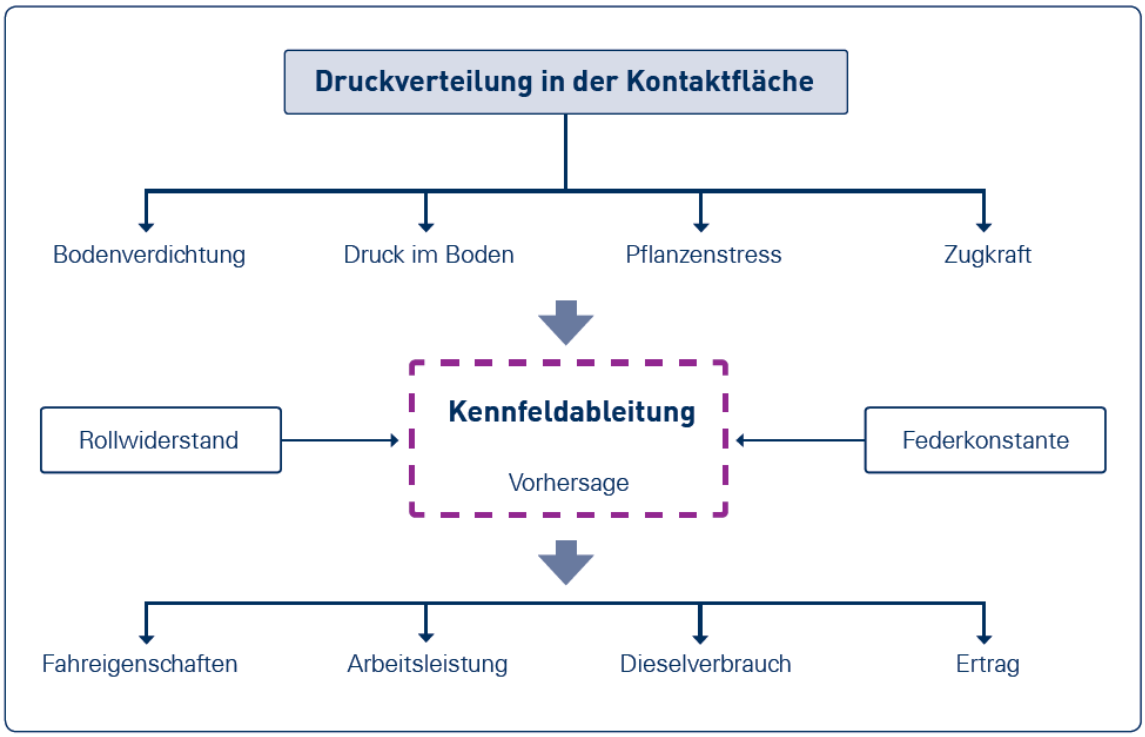


Abbildung 5: Methodik [1]

Inhalt

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Die von Döll, Lehrstuhl Agrarsystemtechnik der Technischen Universität Dresden, entwickelte Methode zur Darstellung agrotechnischer Eigenschaften stützt sich auf die grundlegenden Gedanken von YONG, BEKKER, BURTH, STEINKAMPF, HORN, EHLERT und PETELKAU und enthält Elemente auf der Basis z. T. langjähriger Messergebnisse, die miteinander verrechnet werden. [1]

Mit der Prüfmethode wird eine Trendbeurteilung des Einflusses von landwirtschaftlichen Reifen auf:

- den Boden- und Pflanzenstress
- den Rollwiderstand und das Zugkraft-Schlupf-Verhalten sowie
- das Deformationsverhalten
  einschließlich von Rückschlüssen auf:
    - eine Ertragsbeeinflussung,
    - den Lockerungsaufwand verdichteter Spuren,
    - den Kraftstoffaufwand,
    - den Verschleiß und die Lebensdauer sowie,
    - das Fahrverhalten und die Schwingungsneigung

ermöglicht.

Die Prüfmethode beruht auf der detaillierten Messung (harte Fahrbahn = reproduzierbar) in Abhängigkeit der Radlast und dem Reifendruck. Dabei werden folgende Kennfelder erfasst:

- Druckverteilung in der Kontaktfläche
- der Rollwiderstand
- Reifenverformung unter vertikaler, horizontaler und lateraler Belastung (3D)

Die Kennfelder werden für den Geltungsbereich von Radlast, Reifendruck und Geschwindigkeit lt. Reifenratgeber der Hersteller erfasst.

Rückschlüsse zum Ertrag und zu energetischen Aussagen werden mit Ergebnissen von bodenkundlichen- und pflanzenbaulichen Untersuchungen verrechnet, wie:

- der Druckcharakteristika unterschiedlicher Bodenarten und Wassergehalte (Döll, List, Horn, DIN),
- Ertragsdepression nach Bodenverdichtung (Ehlert, Petelkau, Döll),
- dem Stressverhalten von Kulturpflanzen (Ehlert, Petelkau, Döll) sowie
- den Reib- und Scherwiderständen (Söhne, Tröbner) unterschiedlicher Bodenarten und Wassergehalte.

Die Ergebnisse der Reifenprüfung sind reproduzierbar und detailliert, mit denen Einsatzbereiche und Risiken sicher abgesteckt werden können. Im Vergleich zu Felduntersuchungen lässt sich der Untersuchungsaufwand drastisch senken.

Die Modellierung bekannter Fahrwerke aus den Ergebnissen der Reifenprüfung hat eine hohe Übereinstimmung (3 – 8%) mit der Überprüfung konkreter Fahrzeuge bezüglich der Bodenverdichtung, Ertragsauswirkung, der Zugkraft und dem Kraftstoffverbrauch ergeben.

Somit kann die Methode für die Einschätzung und Bewertung der agrotechnischen Eigenschaften von landwirtschaftlichen Reifen, insbesondere des Einsatzrisikos und der Traktion für Vergleiche genutzt werden. Die Methode bietet beste Voraussetzungen für gegenwärtige Bemühungen zur Produktzertifizierung. Weiterhin unterstützen die Ergebnisse die Modellierung neuer Fahrwerks- und Maschinenkonzepte, was mit einem 600-PS-Traktor mit 8 Rädern nachgewiesen werden konnte und zwei weitere landwirtschaftliche Großmaschinen befinden sich in der Phase der konstruktiven Umsetzung.

Zur Ergebnisdarstellung sind verschiedene Möglichkeiten gegeben, wie zum Beispiel:

- Dateiablage (im Rahmen der Diplomarbeit realisierbare Variante),
- visuelle und tabellarische Darstellung der Daten,
- elektronische Auskunftssysteme für Kennlinien.

2. Stand der Technik

Der Reifenprüfstand ermöglicht es Off-Road-Räder definiert zu belasten und dabei die Druckverteilung in der Kontaktfläche auf harter Fahrbahn, den Rollwiderstand und die Deformation des Reifens zu ermitteln. Für eine definierte Belastung der zu prüfenden Räder sowie die Verfahrung des Tisches sind zwei unabhängige Hydraulikkreise vorhanden. Die Achslast wird durch zwei Hydraulikzylinder realisiert an deren Enden Kraftsensoren angebracht sind. Die Ermittlung der Druckverteilung in der Kontaktfläche erfolgt mit Minikraftsensoren. Dafür ist im Boden ein Messbalken eingelassen, der über 15 Messfühler mit einem Abstand von 10 cm bestückt ist. Um eine Auflösung der Daten im cm-Abstand zu realisieren, muss der Messbalken quer zur Fahrtrichtung (X-Achse), ausgehend von einer definierten Ausgangsposition, 9 mal in 1-cm-Abschnitten verschoben werden um die jeweiligen Messwerte aufzunehmen. In Fahrtrichtung erfolgt die Messwertaufnahme in cm-Schritten durch Anheben und wiederholtes Belasten des Prüfrades und durch Verschieben des Tisches.

Die ermittelten Messwerte werden in eine EXCEL-Tabelle übernommen. Dort wird durch entsprechende Diagramme die konkrete Druckverteilung visuell und tabellarisch dargestellt. Die Messwerte sind einerseits die Grundlage für die Weiterverarbeitung zum Druckabbau im Boden und Bodenkennwerten zur Bodenverdichtung und der Auswirkung auf den Ertrag. Andererseits dient die Druckverteilung mit Druck abhängiger Charakteristik der Reib- und Scherwiderstände zur Bewertung der Traktion.

Bei der Messung des Rollwiderstandes werden nach der Belastung des Prüfrades, die Fahrbahn, ein auf Rollen gelagerter Tisch, unter dem Prüfrad verschoben bzw. verfahren und die Daten belastungsabhängig abgespeichert. Durch die Länge des Tisches von 2 m, sind wiederholte, zusammengesetzte Messabschnitte für einen gesamten Abrollumfang erforderlich.

Für den Rollwiderstand ist am Tisch ein Scherkraftsensor vorhanden. Die Sensordaten werden im DataScan 7010, einem D/A-Wandler für die Industrie, zusammengefasst und über eine serielle Schnittstelle zur weiteren Verarbeitung bereitgestellt. Aktuell werden alle Daten durch den Steuercontroller entgegen genommen und von dort, an den angeschlossenen PC, zur weiteren Verarbeitung bereitgestellt.

Im Rahmen eines „Großen Beleges“ [2] wurde der Reifenprüfsand durch eine studentische Arbeit vom manuellen auf automatischen Betrieb umgebaut. Dabei ist der Messbalkenantrieb auf einen elektrischen Antrieb umgerüstet worden. Für die Positionsbestimmung wurden Endschalter und für die Positionsbestimmung eine Schlitzscheibe vorgesehen. Die Auswertung der Daten für die Steuerung erfolgt im Controller. Für den Controller wurde eine Bedieneinheit entwickelt, die die elektronischen Komponenten aufnimmt.

Die Ermittlung der Daten für die Kontaktfläche des Reifens sowie der Daten des Rollwiderstandes, werden der Tisch und der Messbalken verfahren. Theoretisch sind Bodendrücke für eine Fläche von 31.800 cm² (200 cm Länge, 159 cm Breite) im Zeitmultiplexverfahren möglich. Dabei wird eine Auflösung von 1 cm² bzw. 2,54 Messpunkte/Zoll (dpi), mit einer Positionierung im 1-cm-Raster, realisiert. Die Ermittlung der Tischposition wird durch ein Längenmesssystem auf Basis eines Inkrementalgebers realisiert. Dieser ist zusammen mit den Endschaltern an einem Seilzug befestigt. Die Impulse des Gebers werden im Steuercontroller verarbeitet.

Die gesamte Steuerung ist derzeitig auf Basis eines Mikrocontrollers vom Typ ATmega64 realisiert worden. Für die Ansteuerung der Hydraulik und des Motors für den Messbalken wurden Treiberplatinen in Einzelbaugruppen ausgeführt. Die Software für den Controller ist in C programmiert. Das Programm ist so strukturiert, dass für alle Signale/Baugruppen Treiber geschrieben wurden. Damit sollten Möglichkeiten einer einfachen Erweiterung geschaffen werden.

Die eigentliche Steuerung der Anlage erfolgt über die serielle Schnittstelle. Der PC ist dafür über den USB-Anschluss verbunden. Die Bedienoberfläche auf dem PC basiert auf MatLab.

Im Praxisbetrieb von Reifentests treten jedoch folgende Probleme auf:

- die Kommunikation des PCs mit dem Steuercontroller terminiert häufig mit Timeouts,
- die Messbalkenposition ist nicht gesichert, weder die Ausgangsposition noch die 1-cm-Schritte – die Messergebnisse sind damit nicht weiter verwertbar,
- die Achslast wird manchmal nicht angezeigt, die Gültigkeit der Werte sind nicht gesichert,
- die Sollwerte für die Achslast werden teilweise erheblich überschritten, eine Justierung ist nur manuell an der Drossel möglich,
- Not-AUS ist manchmal ohne Funktion – Gefahr für das Leben – eine Weiterarbeit ist dann erst wieder nach einem kompletten Neustart aller Komponenten möglich,

- Not-Aus löst selbständig aus, sobald ein zusätzlicher Verbraucher an die selbe Steckdosenleiste des Controllers angesteckt wird,
- nach Kommunikationsverlust stürzt die Anwendersoftware auf dem PC ab,
- für die auf MatLab basierende Nutzeroberfläche ist eine Lizenz notwendig um Anpassungen durchzuführen, für die Bedienung reicht die vorhandene Runtime-Version aus,
- der PC ist durch die MatLab-Runtime sehr stark belastet,
- der normale Programmstart der Oberfläche dauert rund 2 ½ Minuten

2.1 Software

Die entwickelte Software von Jaster [2] lässt sich in zwei Bereiche einteilen, die Nutzeroberfläche auf dem PC und die Software des Steuercontrollers.

Die Nutzeroberfläche basiert auf MatLab. Für die Ausführung des Programmes auf dem Bediener-PC ist lediglich eine Runtime, die kostenlos zur Verfügung steht, notwendig. Für Änderungen an der Nutzeroberfläche muss allerdings eine Vollversion eingesetzt werden. Diese Vollversion steht am Arbeitsplatz nicht zur Verfügung. Die Nutzung einer Vollversion ist nur zeitlich und örtlich eingeschränkt im Bereich des TU-Campus möglich. Daraus ergeben sich erhebliche Einschränkungen in Bezug auf mögliche Zugriffszeiten und damit auf die notwendigen Ressourcen, die für Programmänderungen erforderlich wären. Ein weiterer Nachteil besteht darin, dass Änderungen nicht sofort am Prüfstand verifiziert werden können. Bei Fehlern der Implementierung zieht dies erhebliche zeitliche Belastungen nach sich.

Die Software des Steuercontrollers vom Typ ATmega64 basiert auf der Programmiersprache C. Die Entwicklungsumgebung wird durch ATMEL kostenlos zur Verfügung gestellt. Ein Compiler wird unter der GNU-Public-Lizenz angeboten. Damit stehen die Werkzeuge für die Anpassung des vorhandenen Controllerprogrammes zur Verfügung.

Das Programm selbst ist sehr komplex. Der Autor der Software hat die Hardware und alle Baugruppenbereiche in einzelne Module zerlegt und greift anschließend im Hauptprogramm auf die definierten Schnittstellen der Treiber zurück. Es ist eine Eventsteuerung implementiert, die zyklisch nach Einträgen in einer Liste abgefragt wird.

Prinzipiell soll durch die Strukturierung eine Erweiterung des Programmes erleichtert werden. Der Programmcode selbst wurde lediglich mit Doxygen dokumentiert. Diese Dokumentation hat einen Umfang von mehr als 200 A4-Seiten. Dennoch fehlen wichtige Hinweise auf Bezeichner und deren Bedeutung, Logiktabellen, Funktionen und Werte sowie Zusammenhänge in der Steuerung.

Programmablaufpläne, Zustandstabellen, Übersichten, die Zusammenhänge zwischen Programm und elektronischen/elektrischen Komponenten sind nicht dargestellt. Auf Grund der Komplexität sowie der fehlenden Angaben gestaltet sich eine Einarbeitung in die Funktionsweise des Gesamtsystems sowie die Denkweise des Autors als schwierig und sehr aufwändig.

Erschwert wird die Situation dadurch, dass der binäre Inhalt des Steuercontrollers nicht mit dem Inhalt der Binärdatei übereinstimmt, die der Dokumentation beiliegt.

Des Weiteren ist der zur Verfügung stehende Programmcode nicht kompilierfähig. Die Weiterentwicklung der aktuellen Lösung ist somit beinahe unmöglich, da der tatsächlich aktuelle Stand nicht bekannt ist. Eine Pflege des Codes im Controller wäre auf Assemblerbasis durchaus möglich, steht aber in keinem Verhältnis zum zeitlichen Aufwand, zudem ein integriertes Debugging über den eingebauten Programmieradapter nicht möglich ist.

Zwischenzeitlich liegt eine kompilierfähige Version vor, die durch den Autor [2] nachgebessert wurde. Diese Version wurde bisher jedoch nie im Controller getestet. Ein nachträglicher Test bedeutet ein hohes Risiko für den Verlust der vorhandenen Funktionalität des Prüfstandes. Die Fortführung von laufenden Messaufgaben am Prüfstand durch andere Studenten wäre in einem Fehlerfall nicht mehr möglich.

Die auf MatLab basierende Nutzeroberfläche sowie das Programm des Steuercontrollers kommunizieren über die serielle Schnittstelle, die im Steuercontroller auf USB konvertiert ist, miteinander. Damit ist die prinzipielle Zukunftssicherheit für heutige PC-Systeme gewährleistet. Bevor mit der Nutzeroberfläche gearbeitet werden kann, muss die serielle Schnittstelle manuell ausgewählt werden. Anschließend muss die Anweisung zum Aufbau der Kommunikation mit dem Steuercontroller erteilt werden. Die Software für die USB-Schnittstelle ist den Unterlagen beigelegt. Die Funktionsfähigkeit der USB-Treiber für Windows 7 konnte nicht erfolgreich getestet werden.

Der Verbindungsaufbau zum Steuercontroller ist sehr kritisch, wie auch der Autor in seiner studentischen Arbeit beschreibt. Häufig terminiert der Verbindungsaufbau mit einer Timeout-Fehlermeldung der Anwendung. Dieses Problem ist bereits bei der Inbetriebnahme aufgetreten. Der Autor hat im Programm Anpassungen diesbezüglich vorgenommen, wie das Timing-Problem letztendlich behandelt wurde, geht aus der Dokumentation nicht hervor.

Dieser Fehler tritt sehr häufig auf. Oftmals sind 3 – 4 Verbindungsversuche notwendig, um die Kommunikation zum Controller aufzubauen. Der vorausgegangene Messablauf ist damit unterbrochen, so dass der gesamte Messvorgang mit bis zu 1.000 Wiederholungen (Dauer ca. 4 h) neu gestartet werden muss.

Durch Analyse des Programmaufbaus sowie intensiver Tests des Gesamtsystems konnte die Fehlerquelle eingegrenzt werden. Die Ursache liegt in der Datenübernahme der Messwerte vom DataScan 7010 und deren Weiterleitung an den PC. Dieser Gesamtvorgang dauert durch die interne Verarbeitung zu lange.

2.2 Elektronik

Der Aufbau des HMI (**H**uman **M**achine **I**nterface) [2] beinhaltet die Controllerplatine mit den einzeln ausgeführten Treiberplatinen für den Motoranschluss des Messbalkens sowie der Hydraulikventile sowie die Stromversorgung. Der Aufruf von Kommandos zum Heben/Senken des Rades, die Tischbewegung, die Verschiebung des Messbalkens, die Inbetriebnahme und den Notausschalter, sind Drucktaster vorhanden. Die Funktion dieser Bedienelemente ist gegeben, wenn die Verbindung des Steuercontrollers mit der Anwendungssoftware des PCs erfolgreich abgeschlossen wurde.

Von [2] ist die Bedieneinheit miniaturisiert worden. Es stand die Aufgabe die elektronischen Einbauten unter Beibehaltung der Abmessungen völlig neu aufzubauen.

Alle Anschlüsse werden über die Rückseite des Gehäuses zum direkten Anschluss an entsprechende Anschlussklemmen auf den Platinen durchgeführt (siehe Abbildung 6).

Abbildung 6: Rückseite Bedieneinheit gebaut durch [2]

Bei der Durchführung der Leitungen für die Stromversorgung (230 V) und die Leitungen für den Motor und die Hydraulik wurden die Leitungsquerschnitte nicht korrekt berücksichtigt. Die Isolierungen wurden an den scharfkantigen Durchführungen teilweise stark beschädigt, wie Abbildung 7: Zuführung für die Hydrauliksteuerung beispielhaft zeigt.

Abbildung 7: Zuführung für die Hydraulikssteuerung

Auf Grund möglicher Kurzschlüsse, verbunden mit der Gefahr für Mensch und Leben, sowie für einen drohenden Totalausfall des Steuercontrollers, wurde der Reifenprüfstand außer Betrieb gesetzt. Die Durchführungen wurden entsprechend der Leitungsquerschnitte erweitert und die Leitungen nachgesetzt. Erst danach erfolgte eine Freigabe zur weiteren Arbeit am Prüfstand.

Ein prinzipieller Nachteil der Ausführung besteht darin, dass die Leitungen mittels Steckverbinder, die erst nach dem Durchführen der Leitungen durch die Gehäuserückwand angebracht werden können, direkt auf die Treiberplatinen aufgesteckt bzw. verschraubt werden. Für einen rauen Alltagsbetrieb ist das nicht geeignet. Es besteht die Gefahr von Leiterplattenbrüchen, Wartungsarbeiten werden durch diesen Aufbau erheblich behindert.

Die Steuersignalleitungen für die Zählimpulse und der Endschalter werden direkt mit den Kabelenden mittels Klemmverbinder auf der Platine verbunden. Anschlusspläne liegen nicht vor und sind nicht dokumentiert. Auf Grund der sehr kompakten, platzsparenden Bauweise des Gehäuses, ist der Zugang zu den Klemmverbindern erst nach dem Ausbau sämtlicher Platinen möglich. Die Analyse des Gesamtsystems wird zusätzlich durch die Tatsache erschwert, dass nachträgliche Zusatzbeschaltungen für das Entprellen der Endschalter in „Freiluftverdrahtung“ erfolgt sind, wofür es in der Dokumentation keinerlei Hinweise gibt (siehe Abbildung 8).

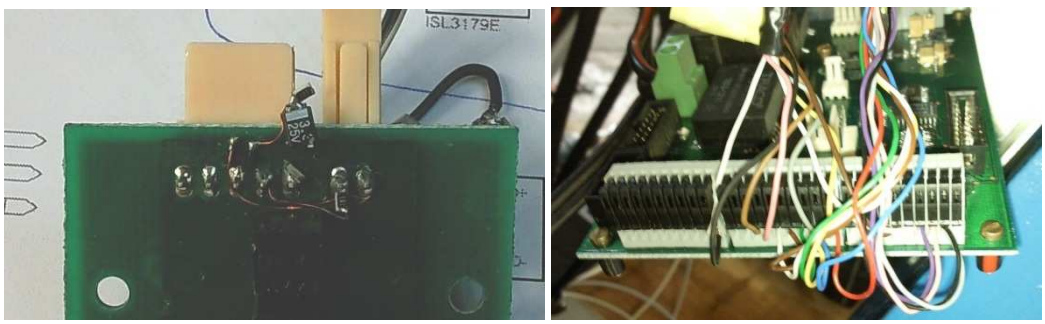


Abbildung 8: Zusatzbeschaltung Messbalken – Endschalter

Abbildung 9: übersichtlich gestaltete Bedienfront [2]

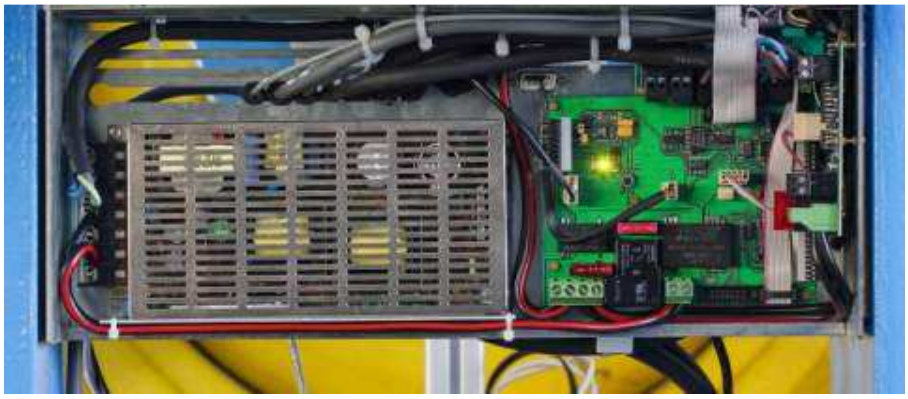


Abbildung 10: Innenansicht des Steuercontrollers [2],
montierte und verkabelte Elektronik in der Bedieneinheit frontal gesehen

Abbildung 11: Blick auf den Motortreiber [2],
montierte und verkabelte Elektronik in der Bedieneinheit

Abbildung 10 und Abbildung 11 dokumentieren die Unterbringung aller elektronischen Komponenten auf engstem Raum.

Die Überprüfung der Elektronik erfolgte an Hand der Schaltungsunterlagen, die der Dokumentation als Anlage beigelegt sind.

Auf Grund fehlender Logiktabellen, Signalpegel/-verläufe, Verdrahtungsplänen des Prüfstandes mussten die Leitungen und deren Signalpegel aufwändig manuell erfasst werden. Durch Einsatz eines Multimeters wurden die Pegel und Signalzustände für die einzelnen Baugruppen ermittelt.

2.2.1 Messbalkenantrieb – Funktionsübersicht

Steckverbinder: J4 Steuercontroller = J1 Motortreiber

Pin	Signal	links	rechts	Bemerkungen
1	--| DGND			DGND
2	PWM-0	5 V	5 V	Power Messbalkenmotor
3	PWM-1	0 V	0 V	Reserve
4	Motor 0D	5 V	0 V	Richtung Messbalken
5	Motor 1D	0 V	0 V	Reserve

Tabelle 1: Belegungstabelle Messbalkenantrieb

An Hand der Messwerte ist erkennbar, dass mit Anlegen des PWM-Signals der Motor eingeschaltet wird. Mit dem Signalpegel Motor 0D wird die Richtung gesteuert, 0 V – rechts und 5 V – links. Die Signale Motor 1D und PWM-1 werden nicht benötigt und dienen als Reserve. Mittels PWM-Signal kann durch Änderung des Tastverhältnisses die zugeführte Energie für den Motor gesteuert werden. Dies wird bei der Ausführung nicht genutzt.

2.2.2 Hydraulikantriebe – Funktionsübersicht

Pin	Signal	Tisch vorwärts	Tisch zurück	Rad auf	Rad ab	Bemerkungen
1 – 4	**Masse**					DGND
5	**Magnet 0**	5 V	0 V	0 V	0 V	Tisch Richtung
6	**Magnet 1**	0 V	0 V	5 V	0 V	Schwinge Richtung
7	**Magnet 2**	0 V	0 V	0 V	0 V	Reserve
8	**Magnet 3**	0 V	0 V	0 V	0 V	Reserve
9	**PWM 0**	5 V	5 V	0 V	0 V	Tisch Ein
10	**PWM 1**	0 V	0 V	5 V	5 V	Schwinge Ein

Tabelle 2: Belegungstabelle Hydraulik

An Hand der Messwerte ist erkennbar, dass mit Anlegen des PWM-0-Signals der Tisch eingeschaltet wird. Mit dem Signalpegel Magnet 0 wird die Richtung gesteuert, 0 V – Tisch zurück und 5 V – Tisch vorwärts. Auf Grund der Beschaltung der Treiberplatine ist darauf zu achten, dass grundsätzlich zuerst der Pegel für die Bewegungsrichtung und erst danach das Powersignal anzulegen sind. Eine Änderung der Richtung bei angelegtem Powersignal führt zu nicht definierten Zuständen bei den Hydraulikventilen, in deren Folge die Betriebsspannung zusammenbricht.

Magnet 1 steuert die Hydraulik des Rades/Schwinge für das anheben – 5 V und absenken – 0 V. Über die PWM-0 und PWM-1 Signale werden die Bewegungen eingeleitet. Mittels der PWM-Signale können durch Änderung des Tastverhältnisses die zugeführte Energie für die Hydraulikventile gesteuert werden. Diese Funktion wird aber nicht genutzt. Die Signale für Magnet 2 und -3 werden nicht benötigt und dienen als Reserve.

Durch externe Stimulierung der Treiberplatinen wurde die Funktionsweise auf Korrektheit der Werte geprüft und bestätigt. Die Treiberplatinen sind somit auch problemlos in Verbindung mit anderen Systemen, die TTL-Pegel führen, einsetzbar. Grundlage für die Steuerung bilden die oben erfassten Logikzustände.

2.3 Messbalken

Der Messbalken muss ausgehend von einer wiederholbaren Ausgangsposition in 1-cm-Schritten bewegt werden können. Dies ergibt sich aus der Anordnung der 15 Minikraftsensoren mit einem Mittenabstand von jeweils 10 cm und dem Wunsch, eine Auflösung der Messwerte von 1 cm² zu erreichen.

Der Messbalkenantrieb ist für einen automatischen Antrieb geeignet. Die Einheit bestehend aus einer Trapezspindel, die durch einen konventionellen Getriebemotor (Gleichstrommotor) angetrieben wird. Für die Positionsbestimmung kommt eine Schlitzscheibe zum Einsatz, die direkt mit der Trapezspindel verschraubt ist. Deren Impulse sind in Verbindung mit der Steigung der Spindel für die 1-cm-genaue Positionierung zuständig. Die Ausgangsposition wird durch einen Endschalter (Drucktaster) angenommen. Unvermeidbare Spindelspiele, beim Wechsel der Bewegungsrichtung, führen damit zwangsläufig zu Positionierungsfehlern.

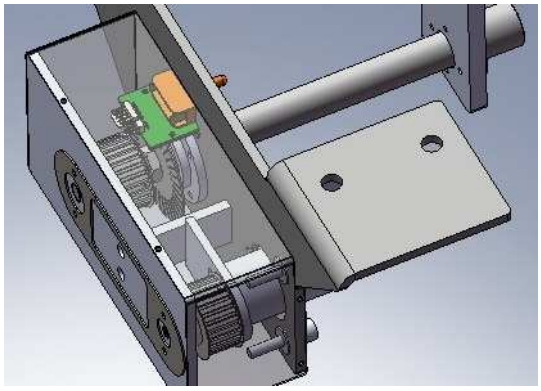


Abbildung 12 : Messbalkenantrieb Modell

[2], Abb. 4.7, S. 14 **[2], Abb. A.23, S. XVII**

Tests des Messbalkenantriebes haben ergeben, dass weder die Ausgangsposition noch die 1-cm-Schritte in der geforderten Genauigkeit eingenommen werden. Die Abweichungen betragen zwischen - 0,2 mm … + 1,3 mm. Gefordert war eine Genauigkeit von 0,3 mm in der Widerholgenauigkeit. Die tatsächlich eingenommenen Positionen wurden in einer Messreihe aufgenommen und werden in Tabelle 3 dargestellt.

Position Richtung	0	1	2	3	4	5	6	7	8	9
Ri → [mm]	51,7	61,6 9,9	72,4 10,8	83,3 9,9	94,2 9,9	104,8 10,6	115,5 10,7	126,3 10,8	136,9 10,6	140,5 Err 3,6
Ri → [mm]	51,5	61,3 9,8	71,9 10,6	82,7 10,8	93,6 10,9	104,8 11,2	115,6 10,8	126,3 10,7	137,1 10,8	140,5 Err 3,4
Ri → [mm]	52,6	62,5 9,9	73,7 11,2	84,4 10,7	95,0 10,6	106,1 10,9	117,0 10,9	128,3 11,3	139,4 11,1	140,6 Err 1,2
Ri ← [mm]	55,0 10,5	65,5 10,9	76,4 10,9	87,1 10,8	97,9 10,8	108,8 10,7	119,3 10,5	129,9 10,6	140,4 10,5	140,6
Ri ←	58,0									140,6

Tabelle 3: Positionen Messbalken

Die Messwerte wurden mittels Messschieber zwischen Gehäuse und Vorderkante der Spindelmutter gemessen. Alle Messungen wurden 3-mal, ausgehend von der Nullposition und 2-mal ausgehend von der Endposition ermittelt.

An Hand der Daten ist erkennbar, dass die Werte in keiner Weise verifizierbar sind. Weder beim Vorgang Messbalken links, noch Messbalken rechts werden die Positionen wiederholt angesteuert. Der Messbalken entfernt sich dabei immer weiter vom Endschalter. Position 9 kann dann gar nicht mehr eingenommen werden, da sich die Spindel bereits am mechanischen Anschlag/Ende befindet. Bemerkt der Bediener diesen Umstand nicht und stoppt den Messbalkenantrieb, kann der Motor durchbrennen. Bei der Dimensionierung des Messbalkenantriebes wurde die Reibung der Spindel und des Messbalkens nicht ausreichend berücksichtigt. Dadurch befindet sich der Antrieb permanent an der Leistungsgrenze.

Ein weiteres Problem besteht darin, dass beim Ausschalten des Controllers bzw. durch eine Fehlfunktion, die zu einem Reset des Rechners oder Controllers führen, die aktuelle Position des Messbalkens zur neuen Nullposition wird.

Die Einnahme der tatsächlichen Nullposition ist nicht mehr möglich. In der Benutzeroberfläche ist die Eingabe negativer Positionswerte für eine manuelle Korrektur nicht vorgesehen. Diese werden als ungültig zurückgewiesen. Der Aufruf der Funktion „Messbalken Rechts“ wird gar nicht ausgeführt, weder am Bedienpult, noch in der Anwendungssoftware. Der Endschalter ist

ohne Funktion. Selbst manuelle Änderungen der Logikpegel, hatten keine Auswirkung. Eine Korrektur der Balkenposition unter den genannten Voraussetzungen ist also nur unter „Ausschluss des Rechtsweges“ möglich, indem man die Position im ausgeschalteten Zustand von Hand herstellt.

Außerordentlich wichtig für einen Vergleich der Reifenkenndaten ist die Position der Drucksensoren. Da diese Daten aktuell nicht verifizierbar sind, können die Messwerte nicht verwendet werden und der Messvorgang muss wiederholt werden. Berücksichtigt man, dass die Erfassung der Messwerte für einen Luftdruck rund einen halben Tag in Anspruch nehmen, so wirkt sich dieser Fehler massiv auf das Ergebnis und die Kosten aus.

Bei der Analyse des Positioniersystems wurde festgestellt, dass die Gabellichtschranke die Schlitzscheibe berührt. Durch Änderung der Befestigung sowie einer Anpassung des konstruktionsbedingten, natürlichen Anschlages der Platine am Gehäuse des Messbalkenantriebes, wurde dieses Problem beseitigt. Auf die Genauigkeit der Positionierung hatte dies keine Auswirkung.

Die Ausführung des Endschalters in Form des gewählten Drucktasters sowie der konstruktionsbedingte Einbau lassen keine Anpassungen an der gewählten Ausgangsposition zu. Ein manueller Nullwertabgleich ist somit nicht möglich. Der Schalter selbst ist als Schließer ausgeführt. Daraus ergibt sich zwangsläufig das Problem, dass bei einer Fehlfunktion oder einem Kabelbruch/-durchtrennung der Messbalken über den Anschlag hinaus gefahren werden kann und dabei den Schalter zerstört. Bemerkt der Bediener dies nicht und bricht den Vorgang ab, kann der Motor durchbrennen.

Die Auswahl des Endschalters ist für den gewünschten Zweck nicht geeignet, da der Weg zwischen Geschlossen und Offen, konstruktionsbedingt, mehrere mm beträgt. Durch Messungen wurden zwischen 3 und 5 mm Hub ermittelt. Die Wiederholgenauigkeit bei dieser Art Schaltern ist nicht gegeben.

Unterhalb der Platine des Endschalters wurde eine elektronische Entprellung ausgeführt. Diese ist durch eine miniaturisierte, in „Freiluftverdrahtung“ ausgeführte Zusatzschaltung auf Basis von SMD-Bauteilen und 0,1 mm CuL-Draht, realisiert worden. In der Dokumentation gibt es keinerlei Hinweise darauf. Bei dieser Schaltung ist ein SMD-Kondensator abgebrochen (siehe Abbildung

8), dessen Bezeichnung nicht zu entziffern ist. Der aus der logischen Funktion erfolgte Ersatz dieses Bauteiles hat wie bereits beschrieben keine Funktion des Endschalters bewirkt.

Die Positionsbestimmung der gewählten Art und in dieser Ausführung ist für den Einsatzzweck nicht geeignet. Als einen der Schwerpunkte der Arbeit sehe ich die Realisierung eines funktionsfähigen Systems für die Positionierung des Messbalkensystems.

2.4 Tischbewegung – Hydrauliksteuerung/Tischposition

Die Bewegung des Tisches in Fahrtrichtung ist durch einen Hydraulikzylinder realisiert. Die Längsposition des Tisches wird durch ein seilgeführtes Längenmesssystem und einen Inkrementalgeber ermittelt. Die jeweiligen Endpositionen wurden durch Reedkontakte aufgebaut. Die am Seil angebrachten Magnete lösen die Reedkontakte aus. Die Verarbeitung der Daten für die Positionen werden im Steuercontroller vorgenommen und in der Nutzeroberfläche angezeigt.

Das Seilzugsystem wurde am Schlitten des Tisches befestigt. Dieser bewegt sich zwischen der Grundfläche und dem Tisch. Die Bewegung des Schlittens gegenüber dem Tisch wird somit auf weniger als die Hälfte reduziert. Der Schlitten befindet sich dabei in nicht wiederholbaren Positionen, sobald die technischen Endanschläge erreicht wurden. Die Endpositionen können auf diese Weise nicht reproduzierbar eingenommen werden.

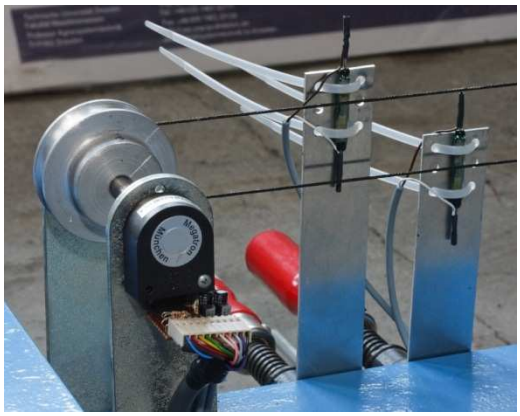


Abbildung 13: Wegmesssystem [2], Abb. A18, S. XV

Die Positionsbestimmung der gewählten Art und in dieser Ausführung ist für den Einsatzzweck nicht geeignet. Als einen der Schwerpunkte der Arbeit sehe ich die Realisierung eines funktionsfähigen Systems für die Positionierung des Tisches.

2.5 Rad – Achslast

Die Belastung-, sowie das Heben und Senken des Rades erfolgt über zwei Hydraulikzylinder, die seitlich an der Schwinge angebracht sind. Das Rad wird dafür in einer Achse über anpassbare Aufnahmen geführt. An den Hydraulikzylindern wird die tatsächliche Belastung des Rades/Reifens mittels zweier Drucksensoren ermittelt. Die Daten werden im DataScan 7010 mit den Daten der Minikraftsensoren im Messbalken zusammengefasst und über die serielle Schnittstelle im Steuercontroller verarbeitet. Von dort werden die Daten über die serielle Schnittstelle/USB an die Nutzeroberfläche des PC übergeben und zur Anzeige gebracht. Über die Nutzeroberfläche kann die gewünschte Achslast eingestellt werden.

Diese Funktionalität ist gegeben. Dennoch kommt es vor, dass die Daten für die Achslast nicht angezeigt und damit nicht verifiziert werden können. Ein Neustart der Software und des Steuercontrollers behebt i. d. R. diesen Fehler. Die Belastung des Rades erfolgt bis zum gewünschten Sollwert. Ist dieser erreicht, wird die Hydraulik ausgeschaltet. Der nachlassende Druck in der Hydraulik wird nicht ausgeglichen. D. h., der Sollwert wird nach einigen Sekunden unterschritten. Für die Ermittlung des aktuellen Bodendruckes ist dieser Zustand vernachlässigbar. Dies begründet sich darin, dass mit Erreichung einer einzustellenden Haltezeit, die Messwerte für den Bodendruck einmalig erfasst und anschließend gespeichert werden.

Bei der Erfassung des Rollwiderstandes erfolgt eine kontinuierliche Datenerfassung des Scherkraftsensors im Tisch über die gesamte Bewegungszeit. Der Tisch kann max. 2 Meter verfahren werden. Bei einer Tischgeschwindigkeit von 0,1 m/s dauert dieser Vorgang 20 Sekunden. Eine regelmäßige Nachbelastung des Rades während des Messvorganges wäre für die Messergebnisse unabdingbar, ist aber nicht realisiert.

Bei manuellen Messaufgaben, die eine konstante Belastung erfordern kann man sich behelfen, indem die Drossel mit viel Geschick so eingestellt wird, dass der Sollwert nicht erreicht werden kann. In Verbindung mit der Einstellung eines geringfügig höheren Sollwertes, kann man beinahe einen Idealzustand erreichen. Einem so eingestellten Sollwert nähert sich das System langsam. Im Automatikbetrieb ist dieser Zustand nicht möglich, da der Messvorgang in Verbindung mit der Wertübernahme erst ausgeführt werden kann, wenn der Sollzustand erreicht wurde.

3. Präzisierung der Aufgabenstellung – Ableitung der weiteren Schritte

- Recherche möglicher Messsysteme für die Positionsbestimmung des Tisches und des Messbalkens, zur erheblichen Vereinfachung der programmtechnischen Anbindung sollte nur ein Verfahren zum Einsatz kommen, bspw. auf Basis von Lichtschranken aus der Automatisierungstechnik
- Analyse des A/D-Wandler Systems zur Datenerfassung/-auswertung mit DataScan 7010 direkt auf dem PC
- Prüfen der Möglichkeiten einer Messwerterfassung und Visualisierung direkt auf dem PC
- Recherche nach möglichen Standardcontrollern, Systemen aus dem Bereich SPS zur Entlastung des Steuercontrollers
- Positionserfassung sowie Endlagenbestimmung unter Einsatz der vorhandenen Sensoren vs. Neuentwicklung
- Beantragung der Freigabe notwendiger Mittel und Beschaffung der Komponenten
- Anfertigung der Längenmesssysteme und Endpositionssysteme, Auf- und Einbau
- Umbau der Steuerelektronik unter Nutzung der vorhandenen Treiberplatinen für Hydraulik und Messbalkenantrieb
- Festlegung und Einbau von Steckverbindern am Controller für alle Signal- und Steuerleitungen
- Schaffung der Voraussetzungen einer direkten Datenübernahme/-verarbeitung der DataScan 7010-Daten im PC-System
- Entwicklung der Nutzeroberfläche, der Steuerung und der Datenauswertung auf Basis höherer Programmiersprachen (C#, .NET) und damit der freien Portierbarkeit auf andere PCs sowie einer einfachen Erweiterbarkeit – Vorbereitung für Systeme mit Windows 7 mit 32 bzw. 64 Bit
- Überlegungen zu den benötigten Daten, der Speicherung sowie für eine externe Weiterverarbeitung in gängige Austauschformate (XML, Text, CSV)
- Festlegung einer zukunftsträchtigen Datenablage und Visualisierung für die Messwerte

4. Vorschlag für Lösungskonzept

- Entwicklung und Austausch der Positionierungssysteme für den Messbalken und den Tisch durch ein einheitliches System, basierend auf Lichtschranken in Verbindung mit einer „Lochleiste“ im 5-mm-Raster
- Endpositionen indem Durchbrüche mit einem ½ Rastermass außerhalb der ersten/letzten Position am U-Profil auf der gegenüberliegenden Seite der Positionen angeordnet sind
- Verarbeitung aller gemessenen Daten vom DataScan 7010 direkt am PC über die serielle Schnittstelle und damit erhebliche Entlastung des Steuercontrollers
- Reduzierung der Softwarearchitekturen auf ein einziges System mit moderner, zukunftsorientierter, objektorientierter Software auf Basis des .NET-Frameworks, C# Version 2010 (in der Express-Version für jedermann frei verfügbar, im Rahmen der MSDNAA für Studenten frei verfügbar)
- Austausch des Steuercontrollers durch einen einfach konfigurierbaren sowie leicht zu erweiternden Controller (Zukauf OpenCockpits [3])
- Konfiguration des Controllers entsprechend der verwendeten Eingabe-/Ausgabeports
- Anwendungsprogramm/Nutzerschnittstelle ist auf der Basis von C# komplett neu zu Programmieren
    - manuelle Steuerung aller Einzelfunktionen sowie der Positionserfassung
    - Automatikbetrieb für Kontaktdruck und Rollwiderstand
    - Messwerterfassung und –speicherung
- Datenexport in CSV/XML
- Vorbereitung der Datenablage in einem, auch zukünftig, frei verfügbaren Datenbanksystem, z. B. auf Basis MS-SQL-Server in der freien Version
- Um- und Einbau des Controllers in das vorhandene Gehäuse
- Ersatz aller Anschlüsse am Controller durch an der Rückseite angebrachte Steckverbindungen
- Stromversorgung mit Gerätestecker und Netzschalter
- Entwicklung und Anfertigung notwendiger elektronischer Baugruppen für den Anschluss der Lichtschranken am Controller, sowie der Ein- und Ausgabeports
- frühzeitige Einbeziehung von Partnern zum Wissenstransfer
- erhebliche Vereinfachung des Systems für Erweiterungen
- keine neuen Funktionalitäten im Rahmen dieser Arbeit

5. Aufbau Hardware

5.1 Sensoren

Für die Erfassung der Messwerte werden Minikraftsensoren von Burster [3] eingesetzt, die seit 20 Jahren absolut fehlerfrei funktionieren. Aus diesem, sowie aus Kostengründen gibt es keine Veranlassung diese Sensoren auszutauschen.

Für die Erfassung des Bodendruckes sind 15 Minikraftsensoren in einen Messbalken eingelassen. Die Radlast wird über zwei parallel geschaltete Drucksensoren erfasst, der Rollwiderstand wird mit einem Scherkraftsensor ermittelt. Die Zusammenführung der Informationen dieser 17 Sensoren bei gleichzeitiger Umwandlung der Messwerte in serielle Daten erfolgt in einem für die Industrie entwickelten D/A-Wandler vom Typ DataScan 7010, wie in Abbildung 14 zu sehen.

Abbildung 14: DataScan 7010

Das DataScan ist durch Zusatzbaugruppen erweiterbar. In maximaler Ausbaustufe können 256 Sensoren angeschlossen und deren Daten verarbeitet werden. Die aktuelle Ausbaustufe verfügt über drei 8-Port Erweiterungen vom Typ 7021 (siehe Abbildung 15). Damit stehen insgesamt 24 Analog Ports zur Verfügung. Davon werden 17 Ports verwendet, weitere 7 Ports können für zukünftige Erweiterungen genutzt werden. Die maximale Datenrate für die Übertragung der Messwerte an einen PC beträgt 38.400 baud. Für 16-bit-Werte können im „continuous scan“ bis

zu 40 Messwerte/Sekunde und Kanal bereitgestellt werden. Dabei werden 4 Byte pro Messwert übergeben. Die typische Messgeschwindigkeit wird mit 1 Hz angegeben.

Abbildung 15: DataScan 7021

Für die Konfiguration der Sensoren sowie die Initiierung von Messvorgängen verfügt das DataScan über 50 integrierte Befehle. Damit ist es möglich die Kommunikation aufzubauen, die Sensoren anzumelden, die Wertebereiche und Skalierungen zu definieren, eine Kalibrierung durchzuführen, die Übertragungsformate festzulegen sowie die eigentlichen Messvorgänge auszuführen. Die Konfiguration bleibt auch nach dem Ausschalten erhalten und entspricht dem Stand beim Ausschalten.

Alle zum Einsatz kommenden Sensoren sind als Full-Bridge-Strain-Messbrücken ausgeführt, die auch dem heutigen Stand der Technik entsprechen. Abbildung 16 zeigt das Anschlussschema.

6.2.4.5 Strain Gauge Connections

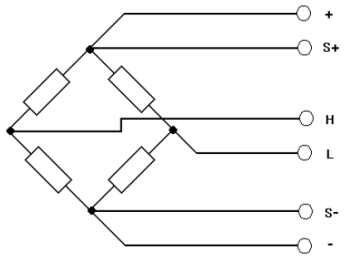


FIGURE 6.16 FULL BRIDGE STRAIN CONNECTION.

Abbildung 16: Full Bridge Strain Connection - DataScan 7010
[4], S. 58

Für eine reibungslose Kommunikation ist die Festlegung des Modus, in dem die Datenübertragung zum Client erfolgen soll wichtig. Damit werden Rückmeldungen, Darstellungsformate und Echos vereinbart.

Bei unklaren Ergebnissen bzw. nicht schlüssigen Messwerten empfiehlt der Hersteller das Rücksetzen des DataScan. Dabei werden alle Einstellungen auf den Auslieferungszustand zurückgesetzt, auch die der angeschlossenen Sensoren. In der Analysephase ist genau diese Situation eingetreten. Es wurden nicht nachvollziehbare Werte angezeigt. Eine Kommunikation zum DataScan konnte nicht mehr hergestellt werden. Aus diesem Grund wurde dieser Vorgang durchgeführt. Es ist durchaus möglich, dass diese Situation erneut auftreten könnte.

Der Hersteller beschreibt folgende Vorgehensweise:

1. aktuelle Einstellungen der DIP-Schalter notieren
2. alle DIP-Schalter auf 1 setzen
3. beim Einschalten sollten alle LEDs kurz blinken, anschließend kann DataScan wieder ausgeschaltet werden
4. alle DIP-Schalter auf die ursprünglichen Einstellungen zurücksetzen
   [4], S. 80

Anschließend müssen die Sensoren erneut angemeldet und eine Kalibrierung durchgeführt werden. Wie dieser Vorgang bei fehlenden Datenblättern bzw. fehlenden Aufzeichnungen erfolgen kann wird im Folgenden beispielhaft für die Bodensensoren beschrieben. Bekannt ist, dass die Sensoren über eine lineare Kennlinie verfügen und als Vollbrücken ausgeführt sind. Dadurch ist es möglich, diesen Vorgang mit geeichten Gewichten bzw. Waagen, durch Vergleichsmessungen

durchzuführen. Der Messvorgang wird dabei über eine Terminalverbindung, per Kommando ausgelöst.

Im Terminal sind dafür folgende Parameter einzustellen:

Datenrate 38.400 Baud, 1 Start-Bit, 8 Datenbits, 1 Stopp-Bit, keine Parität, XON/XOFF.
Als Terminalprogramm eignet sich die Freeware PuTTY. [5]

Für die Kommunikation sind folgende Einstellungen notwendig:

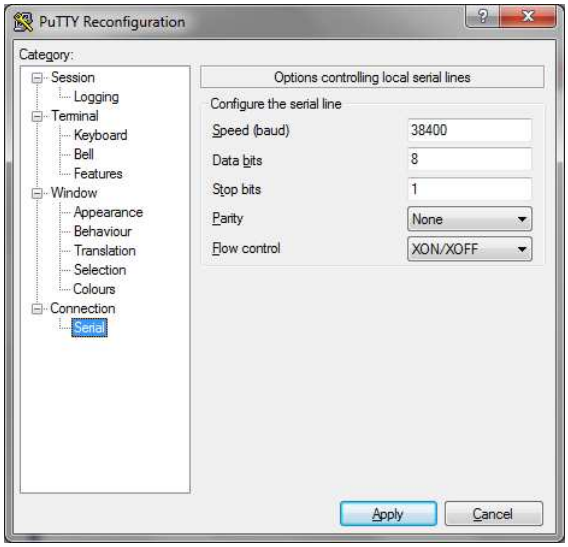


Abbildung 17: Putty - Connection

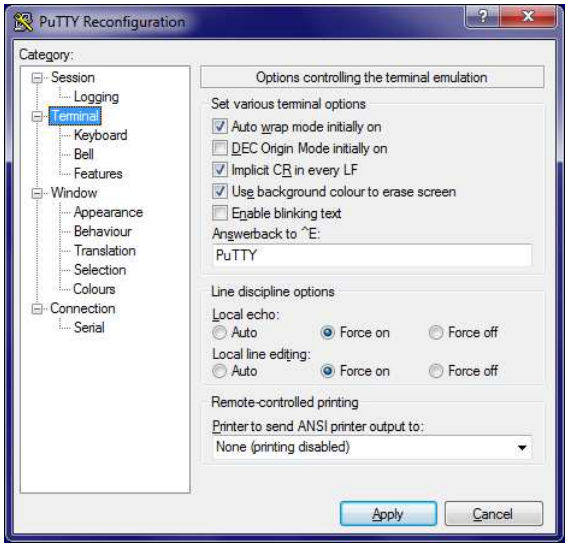


Abbildung 18: Putty - Terminaleinstellungen

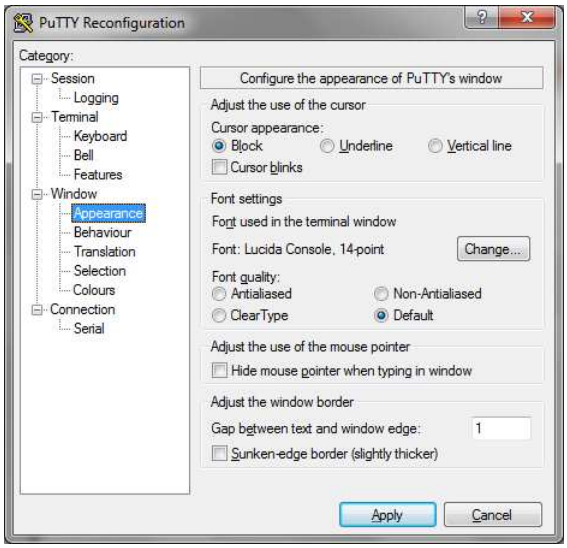


Abbildung 19: Putty - Translation

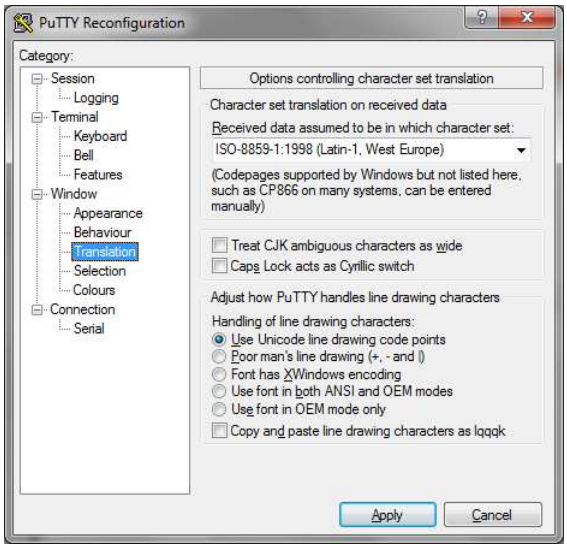


Abbildung 20: Putty - Font settings

Es kann auch jedes andere Terminalprogramm verwendet werden. Die o. g. Einstellungen sind entsprechend zu übertragen.

Der Ablauf dieser Prozedur erfolgt in folgender Reihenfolge:

```
^E            // Aufbau der Kommunikation/Reset mit <Strg> + E
TC            // Controllerstatus erlangen
SS            // Stop Scan, stoppt den continuous scan, wenn eingeschaltet
RM0           // Ausgabeformat auf ASCII im Langformat
CT1,18,16     // Anmelden der Sensoren 1 bis 18 als Full Bridge Strain = 16
CS1,18,0,1    // Sensoren 1 bis 18 auf Autorange = 0 und 16 Bit = 1 setzen
```

Mit folgender Anweisung erfolgt die Kalibrierung der gewählten Sensoren:

```
CZ1,15,8.36   // Initialise Strain Gauges, setzt Kalibrierungsfaktor 8,36
                 für die Sensoren 1 bis 15 = Bodensensoren
RL1,15        // Abfrage der Messwerte für Sensoren 1 bis 15 = Bodensensoren
```

Der erste Lesebefehl nach dem CZ-Kommando setzt die aktuellen Daten auf null. Die Sensoren müssen sich bei Ausführung dieses Kommandos im unbelasteten Ruhezustand befinden, so wie in der folgenden Abbildung 21 zu erkennen ist.

Abbildung 21: Messbalken mit direktem Antrieb und Lochschiene für Positionierung

Im Praxisbetrieb liegen die Messwerte für die Bodensensoren bei maximal 20 kg. Um einen möglichen Linearitätsfehler auszugleichen ist der Einsatz von mehreren Gewichten, die einen möglichst großen Messbereich abdecken, notwendig. Es wurde ein Hammer mit einem gemessenen Gewicht von 1.530 Gramm, entspricht 15,01 Newton, sowie ein Gewicht von gemessenen 20,0 Kilogramm, entspricht 196,2 Newton, verwendet. Für die Ermittlung der Kalibrierungsgewichte wurden geeichte, elektronische Waagen verwendet. Dabei wurde das Prüfgewicht in derselben Form ermittelt, wie auf der Abbildung 22: Kalibrierung Bodensensoren zu sehen ist.

Als Anfangsfaktor für die Kalibrierung wurde vorab ein **Skalierungsfaktor von 2 angenommen**, der ein Überschreiten des intern darstellbaren Zahlenbereiches des DataScan verhindern soll. Das ist notwendig, da ein Überschreiten des Wertebereiches im DataScan zum internen Bereichsüberlauf führt, der alle angeschlossenen Sensoren beeinflusst. Aus diesem Grund müssen auch offene Sensoranschlüsse der 7010er Module auf einen definierten Pegel gebracht werden. Im Handbuch wird empfohlen, diese mit einem vorhandenen Sensor parallel zu schalten.

„9.2.2 Unused inputs – All unused inputs should be set to a defined input. This is best acheived by shorting all poles on a given channel for each unused channel. This helps alleviate floating inputs and range hunting on certain scanner modules.“ [4], S. 82

Abbildung 22: Kalibrierung Bodensensoren
links Hammer mit 1.530 g = 15,01 N, rechts mit 20 kg = 196,2 N auf Unterlegscheibe, die dem Durchmesser der Sensorstößel entspricht

Die durch DataScan ausgegebenen Werte (*Messwert*) werden mit den per Waage ermittelten Werten ins Verhältnis gesetzt. Der daraus resultierende Faktor kann anschließend als Kalibrierungsfaktor eingetragen und dann überprüft werden.

$$\frac{Messwert\ [N]}{Sollwert\ [N]} * aktueller\ Faktor = neuer\ Faktor$$

Die Ermittlung des Kalibrierungsfaktors sollte für den zu erwartenden Messbereich wiederholt und gegebenenfalls erneut berechnet und angepasst werden. Alle Messungen wurden dreifach durchgeführt, da sich die Ergebnisse der letzten zwei Stellen ständig verändern. Für die Betrachtung der Ergebnisse wurden die Mittelwerte verwendet. Bei einem Faktor von 8,3 wurden Abweichungen mit bis zu 3mal dem Vielfachen von 0,17 festgestellt, dies entspricht maximal 0,51 N. Dabei bleibt die Abweichung über den gesamten Messbereich gleich.

Die vorgenommenen Einstellungen des DataScan bleiben auch beim Abschalten im internen Speicher erhalten. Für die Anwendung auf dem PC werden die Daten der ermittelten Kalibrierungsfaktoren übernommen. Dort kann jederzeit die hinterlegte Kalibrierungssequenz ausgelöst werden, sofern dies notwendig ist, bspw. nachdem DataScan in den Fabrikzustand gebracht wurde.

Das folgende Putty-Protokoll sowie die Kommentare dokumentieren den kompletten Vorgang.

```
=~=~=~=~=~=~=~=~= PuTTY log 2011.11.25 13:44:35 =~=~=~=~=~=~=~=~=
$^E
$tc
$ss
$tt01100

// Kalibrierung mit angenommenem Faktor von 2
$cz14,14,2

// Sensoren Nullen, vorhandenen Offset abziehen
$rl14,14
     0.00

// Testmessungen mit 20 kg = 196,2 N Belastung
$rl14,14
   814.30
$rl14,14
   814.64
$rl14,14
   813.79
```

Der Mittelwert aus den drei gemessenen Werten beträgt 814,24. Dieser Wert wird für die Errechnung des tatsächlichen Kalibrierungsfaktors eingesetzt.

$$\frac{814{,}24\ [N]}{196{,}2\ [N]} * 2 = 8{,}3$$

```
// Kalibrierung mit nach obiger Formel errechnetem Faktor von 8,3
$cz14,14,8.3

// Sensoren Nullen, vorhandenen Offset abziehen
$rl14,14
     0.00

// Testmessungen mit 20 kg = 196,2 N Belastung
$rl14,14
   196.52
$rl14,14
   198.22
$rl14,14
   198.22
```

Der Mittelwert aus den drei gemessenen Werten beträgt 197,65. Dieser Wert wird für die erneute Berechnung und Korrektur des Kalibrierungsfaktors eingesetzt.

$$\frac{197{,}65\ [N]}{196{,}2\ [N]} * 8{,}3 = 8{,}36$$

```
// Kalibrierung mit neu errechnetem Faktor von 8,36
$cz14,14,8.36

// Sensoren Nullen, vorhandenen Offset abziehen
$rl14,14
     0.00

// Wiederholung der Testmessungen mit 20 kg = 196,2 N Belastung
$rl14,14
   196.18
$rl14,14
   195.67
$rl14,14
   196.35

// Testmessung mit 1530 g = 15,01 N
$rl14,14
    15.47
$rl14,14
    14.96
$rl14,14
    14.96
```

Die aus den Mittelwerten und den gemessenen Sensorwerten ermittelten Abweichungen betragen bei einer Belastung mit 196,2 N, dem Gewicht mit 20 kg, ein Plus von 0,1%. Bei geringer Belastung mit 15,01 N, Gewicht von 1530 g, tritt ein Fehler von minus 0,8% auf. Der gesamte Messfehler liegt somit unter 1 Prozent. Damit kann der Vorgang der Kalibrierung abgeschlossen werden. Der anzuwendende Faktor für die Minikraftsensoren im DataScan wurde mit 8,36 ermittelt und wird so in die Kalibrierungssequenz der Anwendung übernommen.

```
// abschließende Kontrolle des Sensors im Ruhezustand
rl14,14
    -0.17

// Abschlusssequenz für die Kommunikation mit dem PC
$tt01000
```

Die Kalibrierung für den Scherkraftsensor zur Ermittlung des Rollwiderstandes kann vergleichbar nach o. g. Schema durchgeführt werden. Für die Sensoren der Radlast ist anders vorzugehen. Dort muss berücksichtigt werden, dass sich der Sensor immer im Lastzustand befindet. Im Ruhezustand, wenn das Rad oben ist, wirken bereits die Gewichtskräfte des Rades und der Schwinge in deren Summe. Ist das Rad abgesenkt wirken Gegenkräfte der Hydraulik, die den Sensor am Hydraulikzylinder beeinflussen. Diese Kräfte müssen berücksichtigt werden und werden als ne-

gative Werte ausgegeben. Für solche Fälle ist im Befehlssatz des DataScan die Anweisung „CB“ vorgesehen.

„CBf,l,m,c – Calibration Values
Calibrate channels first (f) to last (l) with scale (m) and offset (c). This command can be used with analog and counting channels.

y = mx + c

Where ‘x’ is the measured value and ‘y’ is the result returned.“ [6], S. 17

Ein Einfaches nullen der Sensordaten im Ruhemodus ist nach Ausführung des „CB“-Kommandos mit „RL17,17“ nicht möglich. Dies muss in der Anwendungssoftware berücksichtigt werden.

Eine dynamische Ermittlung des Kalibrierungsfaktors gestaltet sich dadurch etwas schwieriger, da gesichert werden muss, dass keine Überlastung des Gesamtsystems bestehend aus dem Rad, des verwendeten Reifens und der Schwinge erfolgen darf. Die vorgesehene Auflast, die in der Bediensoftware eingestellt wird, vergleicht den gemessenen mit dem eingestellten Wert. Da der korrekte Kalibrierungsfaktor noch nicht bekannt ist, kann dies zu erheblichen Abweichungen zum tatsächlichen IST führen.

Der Faktor wurde nach o. g. Rechnung ermittelt, wobei auf das Gesamtgewicht des Systems im Ruhezustand geschlossen wurde. Bekannt ist lediglich, dass sich der Sensor linear verhält.

Um die korrekten Werte zu verifizieren wurden mehrere Vergleichsmessungen durchgeführt, die einen Kalibrierungsfaktor von 17 bestätigen. Für die Vergleichsmessungen wurde das Rad auf eine Radlastwaage abgesetzt. Die Anweisung wird mit folgender Anweisung an DataScan übergeben:

```
CB17,17,17,660    // Sensor von 17 bis 17, Faktor 17, Offset 660
```

Nachfolgende Tabelle 4 stellt die gemessenen Werte für verschiedene Belastungsstufen gegenüber.

Radwaage [kg]	Radwaage [N]	Sensordaten [N]	Abweichung
0,1	0,981	0	
890,6	8736,79	8830	1,1%
1139,9 [1]	11182,42	11300 [1]	1,0%
1390,8	13643,75	13810	1,2%
2559,4	25107,71	25440	1,3%
3531,1	34640,09	34900	0,7%

Tabelle 4: Vergleichsmessung Radlastsensor

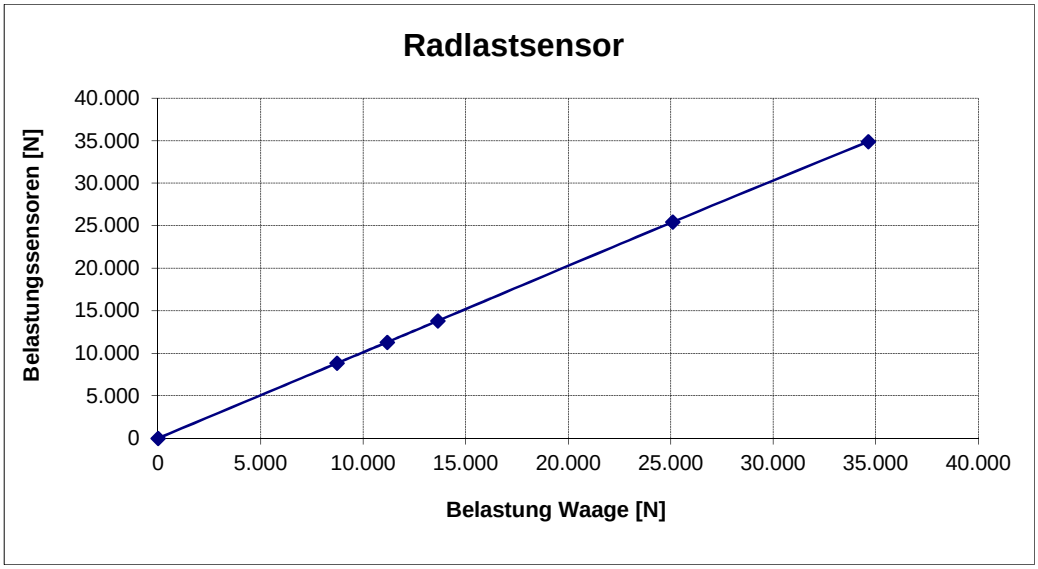


Abbildung 23: Diagramm Kalibrierung Radlastsensor

Die Abweichungen zwischen den beiden Messsystemen beträgt rund 1%. Wie in Abbildung 23: Diagramm Kalibrierung Radlastsensor zu erkennen ist, liegt eine Übereinstimmung der Daten vor. Damit kann der Kalibrierungsfaktor im DataScan eingetragen und die Kalibrierung abgeschlossen werden. Zusätzlich erfolgt die Übernahme des Wertes für die Kalibrierungssequenz der Anwendung. Gut zu erkennen ist der lineare Verlauf der Kennlinie.

Messfehler können nicht ganz ausgeschlossen werden. Dies begründet sich schon darin, dass die Messdaten der elektronischen Radwaage viel schneller aktualisiert werden, als die der Radlastsensoren. Aus diesem Grund wurden die Messungen mehrfach wiederholt. Um Ablesefehler zu minimieren wurden während des Messvorganges digitale Serienbilder angefertigt, auf denen

[1] [2] Abbildung 24: Vergleichsmessung Radwaage - Sensordaten in der Anwendung

gleichzeitig beide Werte abzulesen sind, wie auf Abbildung 24 zu sehen ist. Die Radwaage wurde vor jedem Messvorgang auf null gesetzt. Der Belastungszustand des Prüfrades durch die Hydraulik ändert sich während des Messvorganges ständig. Dass ist Ursache der Streuung der Messfehler, die nicht vom Belastungsgrad abhängig sind (siehe Abbildung 25).

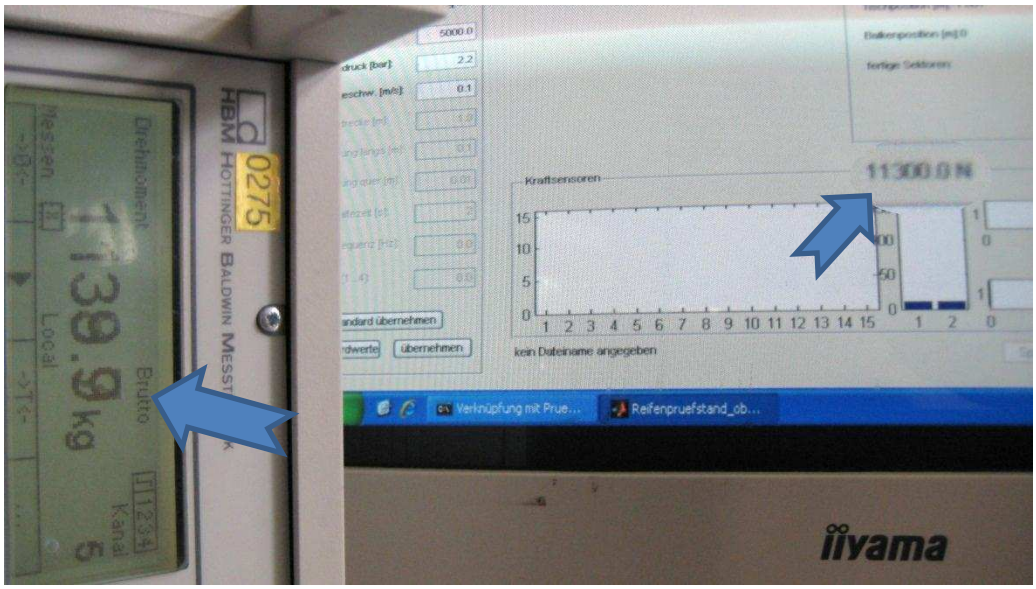


Abbildung 24: Vergleichsmessung Radwaage - Sensordaten in der Anwendung

Folgende Abbildung 25, zeigt die Messfehler im Diagramm.

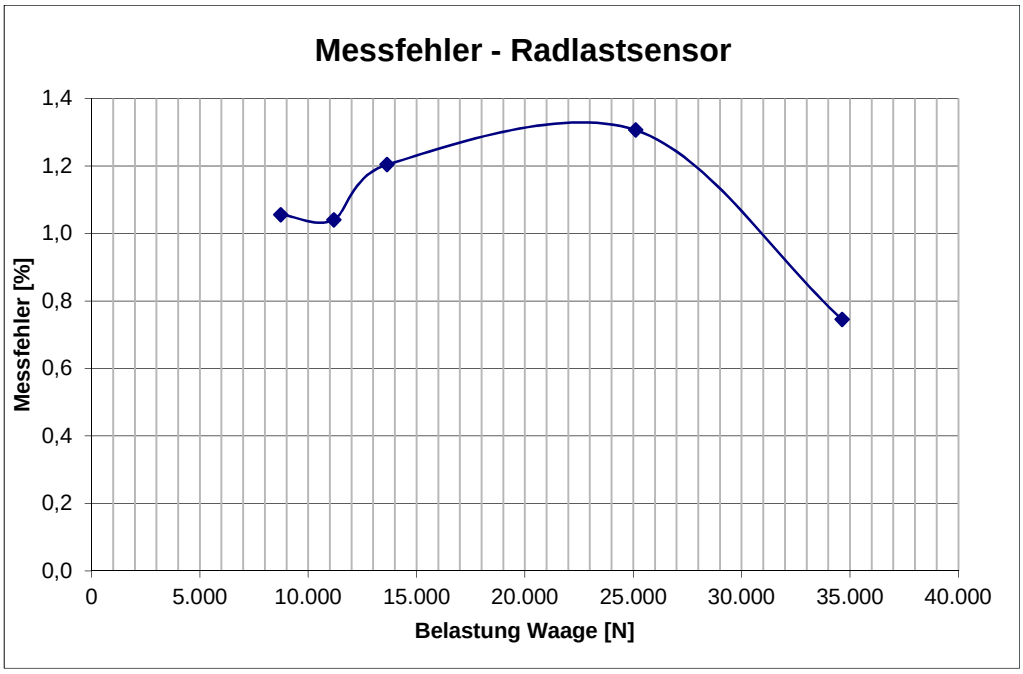


Abbildung 25: Diagramm Messfehler - Radlastsensor

5.2 Systeme zur Positionsbestimmung

Wie bereits bei der IST-Analyse festgestellt wurde, eignen sich die eingesetzten Messsysteme für die Positionsbestimmung nur bedingt für die reproduzierbare Einnahme der gewählten Positionen, sowohl für den Messbalken als auch für den Tisch.

Folgende Tabelle stellt mehrere Möglichkeiten gegenüber:

System	Genauigkeit	Kosten	Zuverlässigkeit	Programmierbarkeit	Erweiterbarkeit
Vorgabe	10 ± 0,3 mm	preiswert	zuverlässig	einfach	einfache Erweiterbarkeit
Schlitzscheibe	genau	preiswert	fehlerträchtig	schwierig	möglich
Inkrementalgeber	sehr genau, 512 … 1024 Impusle pro Umdrehung	mittelmäßig	gut	schwierig	möglich
analoge Messsysteme	sehr genau	sehr teuer	zuverlässig	schwierig, Analogwertverarbeitung	unmöglich
Lichtschranke	optimal an den Bedarf anpassbar	preiswert	zuverlässig	sehr einfach	durch Einbringen zusätzlicher Durchbrüche

Tabelle 5: Gegenüberstellung Positionsbestimmung

Die Entscheidung fällt für eine Version mit Lichtschranke. Das System ist einfach zu beherrschen. Die Positionsbestimmung erfolgt vollkommen unabhängig vom eingesetzten Antriebssystem und deren Geschwindigkeit. Die einzig an das Antriebssystem geknüpfte Bedingung betrifft den Stoppvorgang. Die nach Auslösung des Stoppvorganges zurückgelegte Strecke muss kleiner, als die Hälfte eines Positionsschrittes sein. Im konkreten Fall also kleiner als 5 mm.

Die programmtechnische Umsetzung ist einfach zu realisieren. Eine Anpassung an geänderte Bedingungen, wie z. B. bei einer Verlängerung der Fahrstrecke des Tisches ist lediglich das Profil mit weiteren Durchbrüchen zu versehen und die Endmarken neu festzulegen. Im Anwendungsprogramm ist lediglich eine Änderung der Gesamtlänge vorzunehmen. Dies erfolgt an einer einzigen Stelle. Eine Änderung des Bewegungssystems ist, sofern die elektronischen Treiberplatinen

eingesetzt werden, durch einfachen Anschluss an die vorgesehenen Ausgänge der Treiberplatinen möglich.

Die Genauigkeit der Positionen ist lediglich von der Genauigkeit der Durchbrüche für die Lichtschranken sowie von der Nachlaufstrecke des Systems abhängig. Einflüsse bspw. auf Grund des Spindelspiels bei Änderung der Bewegungsrichtung spielen keine Rolle. Damit ist die Voraussetzung für eine sehr hohe Wiederholgenauigkeit bei gleichzeitig einfacher Handhabung gegeben.

5.2.1 Umrüstung Positionssystem Messbalken

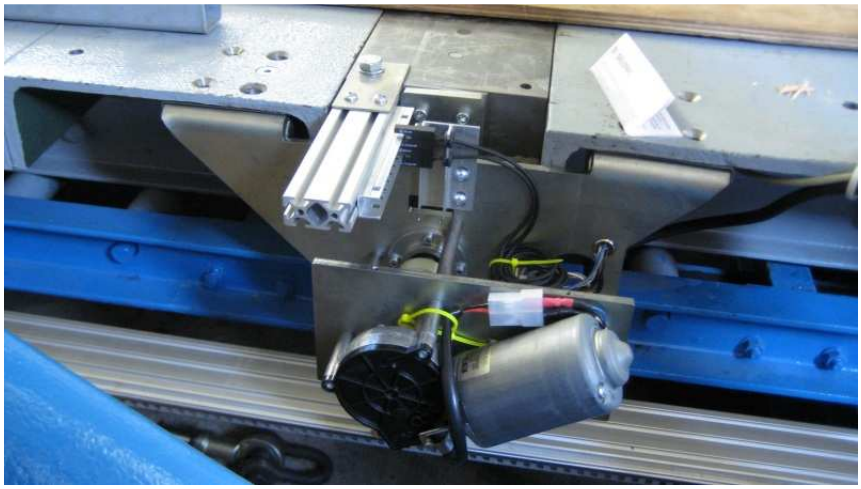


Abbildung 26: Messbalken mit direktem Antrieb über Winkelgetriebe

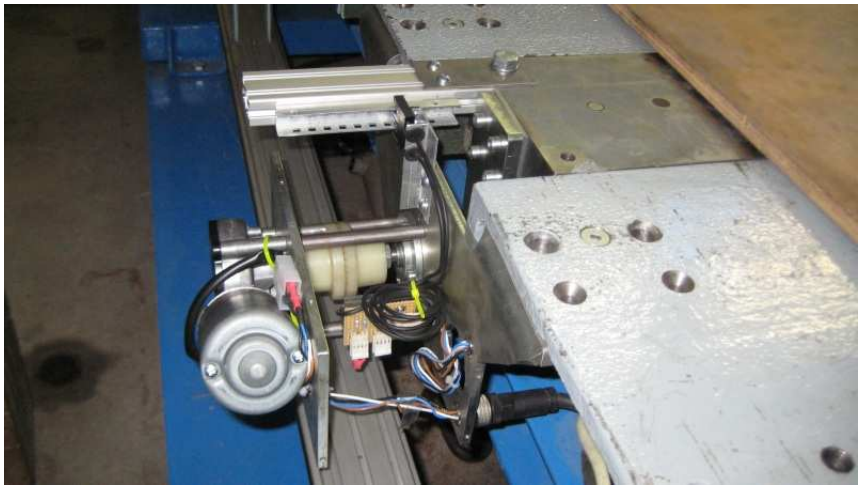


Abbildung 27: Messbalkenantrieb mit Positionssystem, Lochleiste und Lichtschranken

Für die Positionierung des Messbalkens kommt ein U-Profil aus Aluminium zum Einsatz. In dieses Profil wurden für die Positionen 10 quadratische Durchbrüche mit einer Größe von 5 mm und einem Abstand von 1 cm eingebracht. Auf der gegenüberliegenden Seite wurden ausgehend von

der ersten bzw. letzten Position dieselben Durchbrüche mit einem seitlichen Versatz von 2,5 mm eingebracht.

Je nach Bewegungsrichtung werden an den Kanten der Durchbrüche die Lichtschranken geöffnet bzw. geschlossen. In Verbindung mit der Anwendungssoftware werden diese Informationen für die Positionsermittlung genutzt.

Die Befestigung des U-Profils erfolgt direkt am Messbalken, an einem stabilen Aluminiumprofil. Damit wird die direkte Position des Messbalkens ermittelbar. Umrechnungen entfallen und ein mögliches Spindelspiel, wie es insbesondere bei einem Richtungswechsel überwunden werden muss, wirken sich nicht mehr aus. Der Messbalkenantrieb selbst wurde vom System der Positionierung komplett entkoppelt. Bei Bedarf kann das Antriebssystem jederzeit durch ein anderes Antriebssystem ersetzt werden, was auch erfolgt ist. Der Getriebemotor mit Zahnriemen (siehe Abbildung 12) [2] wurde durch einen stärkeren Getriebemotor, welcher die Spindel direkt antreibt ausgetauscht (siehe Abbildung 26 und Abbildung 27).

Die Position der Lichtschranke zum U-Profil kann durch horizontale und vertikale Verschiebung der Lichtschranken auf ihrem Träger erfolgen. Die Positionierung der Endpunkte und damit verbunden der Position des Messbalkens zum Tisch ist durch Verschiebung des U-Profils auf dem Träger möglich.

5.2.2 Umrüstung Positionssystem Tisch

Das Positionssystem für den Tisch wurde nach demselben Prinzip wie für den Messbalken entwickelt. Die Anzahl der Durchbrüche für die Positionen musste, auf Grund technologischer Gesichtspunkte, um einen Zentimeter auf 199 cm verkürzt werden. Die Durchbrüche wurden im Laserzentrum der TU Dresden angefertigt. Die maximale Bearbeitungslänge derer Maschinen beträgt maximal 2 Meter. Die Positionsbestimmung musste somit auf maximal 198 cm verkürzt werden. Dieser Vorgang wurde mit dem Auftraggeber besprochen und akzeptiert.

Genau wie beim Messbalken befinden sich auf der gegenüberliegenden Seite mit einem seitlichen Versatz die Durchbrüche für die Endpositionen.

Abbildung 28 und Abbildung 29 zeigen das System in den Grundbestandteilen.

Abbildung 28: Tischpositionierung mit Lochschiene und Lichtschranken

Die Befestigung des U-Profils erfolgt an einem Trägerprofil mit Rollschienen. Dieses Trägerprofil wurde am Tisch befestigt. Damit wird die Position des Tisches direkt ermittelbar, Umrechnungen entfallen. Der Tischantrieb ist vom System der Positionierung komplett entkoppelt. Bei Bedarf kann das Antriebssystem jederzeit durch andere Komponenten ersetzt werden.

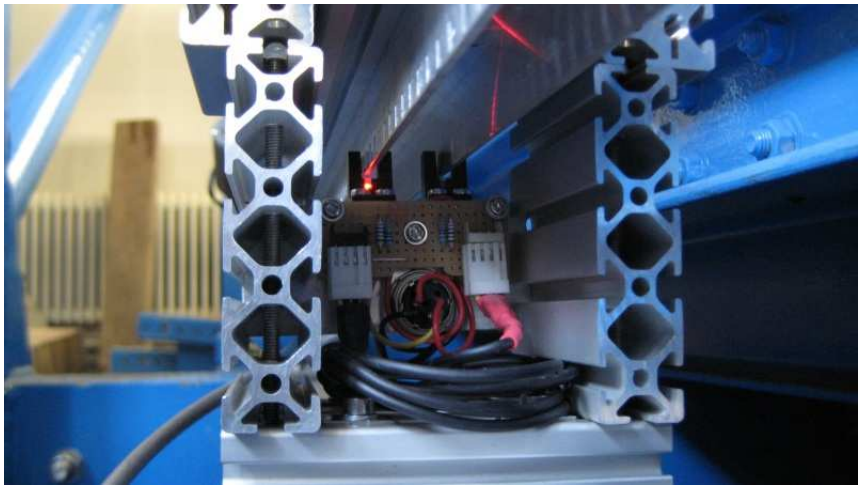


Abbildung 29: Tischpositionierung Lichtschrankenträger

Die Lichtschranken wurden an einen Träger angebaut, der eine einfache Justierung zulässt. Der gesamte Träger kann in Längsrichtung zur Festlegung der Mittenposition, bei 99 cm eingestellt werden. Dazu wird ein Lot an der Radachse befestigt, welches durch den Mittelpunkt der Achse verläuft und dann optisch in die Fluchtlinie der Minikraftsensoren im Messbalken gebracht wird.

Durch das Schienensystem und deren Führung im Lichtschrankenträger wurde eine sehr genaue Führung des U-Profils zu den Lichtschrankenöffnungen realisiert. Seitliche Tischbewegungen und Höhenschläge wirken sich in keiner Weise auf den Abstand der Schiene zur Lichtschranke aus. Einem möglichen Höhenschlag wird dadurch entgegengewirkt, dass der Lichtschrankenträger auf einem in der Höhe beweglichen Träger mit dem Grundträger des Prüfstandes verschraubt wurde. Damit ist ein sicheres, wiederholbares Ansprechen der Lichtschranken gewährleistet.

Die Position der Lichtschranke zum U-Profil kann durch horizontale und vertikale Verschiebung der Lichtschranken erfolgen. Damit ist eine Justierung zum Lochprofil möglich.

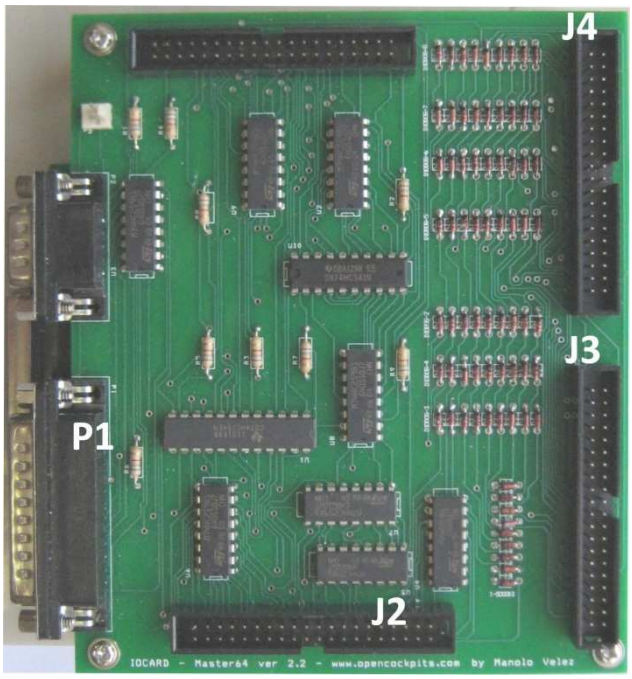
5.3 Steuercontroller

Bei der Recherche zu dieser Arbeit ist der Autor auf einen Artikel in der Fachzeitschrift für Programmierer, der dotnetpro „Das Profi-Magazin für Entwickler“, Ausgabe 5/2010, von Klaus Aschenbrenner aufmerksam geworden. [7], S. 16 ff, „.NET steuert Hardware“

Im Artikel werden Möglichkeiten der Hardwareprogrammierung vorgestellt. Dort werden HardwareBoards der Firma OpenCockpits vorgestellt und beschrieben. Die Boards sind preiswert, eine MasterCard ist schon für ca. 53 EUR erhältlich und bietet bereits in der Grundausstattung umfangreiche Anschlussmöglichkeiten. An eine Master-Card lassen sich 38 digitale Ausgänge, 72 digitale Eingänge sowie 4 DisplayCards anschließen.

Für den Betrieb des Prüfstandes werden derzeit 14 Eingänge und 11 Ausgänge benötigt. Es bleiben somit viele Erweiterungsmöglichkeiten.

Die folgende Abbildung zeigt den prinzipiellen Aufbau einer MasterCard.



P1 = Anschluss einer USB Expansion Card

J2 = Ausgabe Steckverbinder

J3 = Eingabe Steckverbinder

J4 = Eingabe Steckverbinder, wie J3.

Abbildung 30: OpenCockpits – MasterCard

Masse	Out 12	Out 14	Out 16	Out 18	Out 20	Out 22	Out 24	Out 26	Out 28	Out 30	Out 32	Out 34	Out 36	Out 38	Out 40	Out 42	Out 44	Out 46	Out 48
2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40
1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31	33	35	37	39
n. belegt	Out 11	Out 13	Out 15	Out 17	Out 19	Out 21	Out 23	Out 25	Out 27	Out 29	Out 31	Out 33	Out 35	Out 37	Out 39	Out 41	Out 43	Out 45	Out 47

Tabelle 6: OpenCockpits – Pinbelegung Ausgänge (J2)

Input 02	Input 03	Input 07	Input 06	**Masse 1**	Input 11	Input 12	Input 16	Input 15	**Masse 2**	Input 20	Input 21	Input 25	Input 24	**Masse 3**	Input 29	Input 30	Input 34	Input 33	**Masse 4**
2	4	6	8	**10**	12	14	16	18	**20**	22	24	26	28	**30**	32	34	36	38	**40**
1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31	33	35	37	39
Input 01	Input 04	Input 08	Input 00	Input 05	Input 10	Input 13	Input 17	Input 09	Input 14	Input 19	Input 22	Input 26	Input 18	Input 23	Input 28	Input 31	Input 35	Input 27	Input 32

Tabelle 7: OpenCockpits – Pinbelegung Eingänge (J3)

Die Zählweise der Ausgabeports beginnt mit 11 am Pin 3 des Steckverbinders. Pin 1 führt laut Datenblatt + 5 V, dies ist jedoch nicht korrekt, weil es nicht beschaltet ist. Alle Ports nutzen eine gemeinsame Masse.

Eine Besonderheit bei den Eingängen besteht darin, dass diese in 4 Gruppen zu je neun Eingängen mit einer dazugehörenden Gruppenmasse versehen sind. Dies muss bei der Planung der Anschlussbelegung für eigene Anwendungen berücksichtigt werden. Die Belegungstabellen sind sowohl in der Dokumentation von OpenCockpits [8], S. 4 (zwischen den Eingabeports 0 und 4 ist Port 8 und nicht Port 6), als auch im Beitrag in der dotnetpro [7], S. 17 (die Eingabeports wurden komplett fehlerhaft durchnummeriert), nicht richtig dargestellt.

Im Forum [9] wurde mehrfach über scheinbar defekte MasterCards berichtet. Aus diesem Grund wurde durch den Autor ein Testboard zur Funktionsprüfung der MasterCard entwickelt, welches alle 38 Ausgänge als LEDs sowie 36 Eingänge auf Basis von Mikrotastern enthält. Durch umstecken des Steckverbinders von J3 auf J4 kann auf diese Weise eine MasterCard komplett überprüft werden. Im konkreten Fall ist festgestellt worden, dass der Eingabeport 01 nicht sicher funktioniert. Aus diesem Grund wurde dieser Port bei der weiteren Entwicklung nicht belegt.

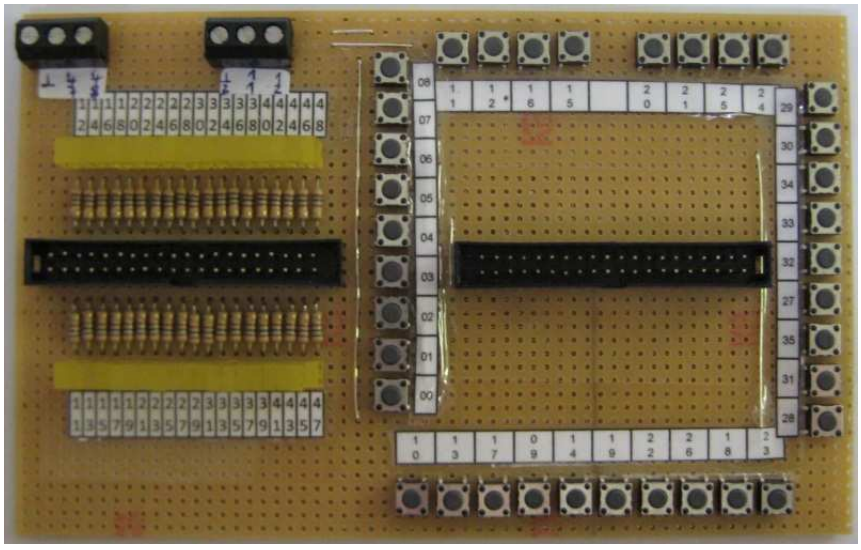


Abbildung 31: Testboard für MasterCard

5.3.1 Stromversorgung

Für die Stromversorgung der Master- und USB-ExpasionCard werden 5 V, für die Hydraulikventile und den Antriebsmotor des Messbalkens 12 V benötigt. In diesem Zusammenhang kann ein normales Computernetzteil eingesetzt werden. Auf Grund der Baugröße sowie der technischen Parameter wurde das FSP Fortron 180-50LE 180W Flex ATX PC-Netzteil [10] ausgewählt. Dieses Netzteil bei 5 V mit 16 A und bei 12 V mit 14 A belastbar.

Der Nennstrom des zu versorgenden Antriebsmotors ist mit 4 A angegeben. Der tatsächlich gemessene Strom bei maximalem Drehmoment beträgt 3,2 A. Die typische maximale Nennspulenbelastung bei den Hydraulikventilen beträgt 2,5 A je Spule. Für die elektrischen Parameter ist somit eine Reserve von 5 A vorhanden, wenn gleichzeitig der Antriebsmotor und zwei Hydraulikventile bedient werden. Dieser Zustand kommt aber in der normalen Praxis nicht vor.

Mit 5 V werden lediglich die MasterCard, die USB-ExpansionCard sowie die LEDs der Statusanzeigen und der Lichtschranken gespeist. In der Summe werden maximal 500 mA benötigt, wenn alle Anzeigen gleichzeitig eingeschaltet sind.

Von der Bauform ist das Netzteil sehr kompakt ausgeführt und lässt sich somit gut in das Gehäuse integrieren. Alle nicht benötigten Anschlüsse wurden durch den Autor entfernt.

Bei Verwendung von Computernetzteilen ohne Mainboard ist zu berücksichtigen, dass diese i. d. R. mindestens eine Grundlast am 5 V-Anschluss benötigen, da ansonsten das Netzteil nicht einschaltet. Zusätzlich müssen Pin14 (grün) und Pin15 (schwarz) kurzgeschlossen werden. Die Netzspannung wird über eine an der Geräterückseite angebrachte Kaltgerätedose und einen Netzschalter zugeführt.

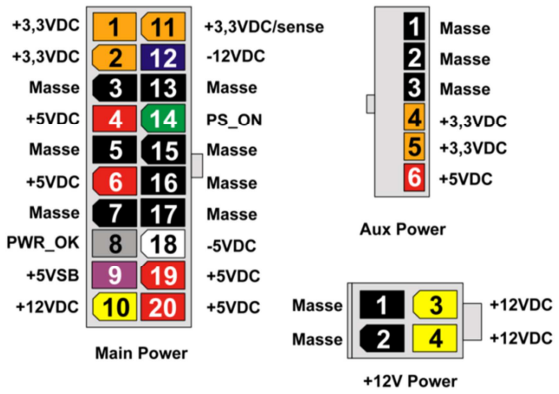


Abbildung 32: Belegung der Stecker eines ATX 1.3-Netzteils [11], http://de.wikipedia.org/wiki/ATX-Format

5.3.2 Umrüstung der externen Anschlüsse – Steckverbinder

Für alle Anschlüsse an der Bedieneinheit (siehe Abbildung 37: Geräterückseite mit Schalter und Steckverbindern) werden Miniatur-Rundsteckverbinder/-buchsen der Serie 581 und 680 von Binder [10] vorgesehen. Diese verfügen über eine Schraubverriegelung, bieten gute EMV-Eigenschaften und sind in der 3poligen Ausführung mit 7 A, in der 5poligen Ausführung mit 6 A belastbar. Die fünfpolige Ausführung wird für die Gabellichtschranken der Positioniersysteme verwendet und damit mit maximal je 40 mA belastet. Für den Anschluss des Getriebemotors (4 A), der Hydraulikventile (je 2 x 2,5 A) und den Endschalter der Schwinge (weniger als 20 mA) werden die 3poligen Ausführungen verwendet. Die Steckverbinder verfügen somit über gute Eigenschaften und genügend Reserven für die Strombelastung. Das versilberte Kontaktmaterial gewährleistet eine hohe Kontaktsicherheit auch bei geringer Belastung.

5.3.3 Auswahl der Lichtschranken für die Positionierung

Für das Positioniersystem wurde ein System auf der Basis von Gabellichtschranken und Lochschienen entwickelt. Bei den Gabellichtschranken kommen Miniaturlichtschranken Serie GL5 der Firma Pepperl & Fuchs [12] zum Einsatz, die bei Conrad Electronic [10] beziehbar sind. Auf Grund der ausgezeichneten elektrischen Eigenschaften sowie einer Ausführung mit Anschlusskabel bei einem angemessenen Preis werden diese zur Nutzung ausgewählt.

Auf der folgenden Abbildung 33: elektrische Beschaltung der Lichtschranke ist zu erkennen, dass die Lichtschranke eine Masse sowohl für die LED als auch für die Auswertung des Zustandes be-

sitzt. Für den konkreten Einsatz wird die Hellschaltung verwendet, das bedeutet, ein High-Pegel signalisiert eine offene, ein Low-Pegel eine unterbrochene Lichtschranke. Die Lichtschranke verfügt über eine integrierte Kontroll-LED, damit ist eine einfache Sichtkontrolle der Funktionsfähigkeit sowie eine spätere Justierung möglich.

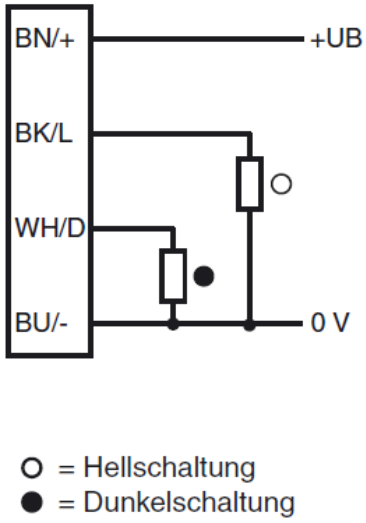


Abbildung 33: elektrische Beschaltung der Lichtschranke

In Verbindung mit dem Anschluss an eine MasterCard und der speziellen Eigenheit der Gruppenmasse für jeweils 9 digitale Eingabegeräte, entsteht somit ein Problem, da die Eingänge der 8 Gruppen (J2 und J3) mit gleichem „Offset“ parallel geschaltet werden. Für die 4 eingesetzten Lichtschranken zieht dies den Verzicht auf 28 von 72 Eingabeports nach sich. Dies ist aber ohne Probleme vertretbar, da bei dem derzeitigen Entwurf lediglich 14 Eingabeports benötigt werden. Somit stehen trotzdem noch 30 Eingabeports für spätere Erweiterungen zur Verfügung.

5.3.4 Anschlussbelegung Ein-/Ausgänge an der MasterCard

Auf Abbildung 34: Eingänge – Anschlussbelegung an J3 ist die Belegung der Taster, Endschalter und der Lichtschranken dargestellt. Die Lichtschranken (LS) und der Endschalter (ES) der Schwinge sind über die I/O-Platine an die Steckverbinder der Gehäuserückseite angeschlossen. Die Taster (TA) wurden intern verdrahtet. Die Farbangaben betreffen die verwendeten Kabelfarben.

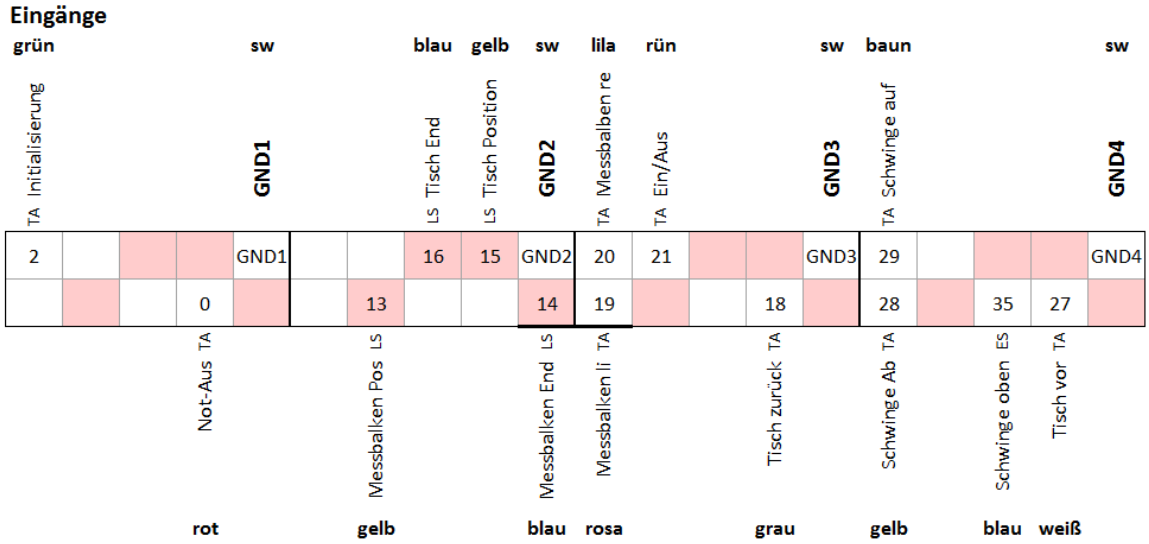


Abbildung 34: Eingänge – Anschlussbelegung an J3

In Gruppe 2 wurde der Anschluss der Gabellichtschranken vorgenommen. Die farblich gekennzeichneten Anschlüsse der Gruppen 1, 3 und 4 wurden aus o. g. nicht verwendet.

Auf Abbildung 35: Ausgänge – Anschlussbelegung an J2 ist die Belegung der Ausgänge dargestellt. Die Motorplatine [2] wird an einem 5poligen Steckverbinder der die Verbindung zur I/O-Platine realisiert angeschlossen. Der Anschluss der Hydraulikplatine [2] erfolgt an einem zweireihigen, 10 poligen Steckverbinder.

Die LEDs wurden, wie auch die LEDs, intern verdrahtet.

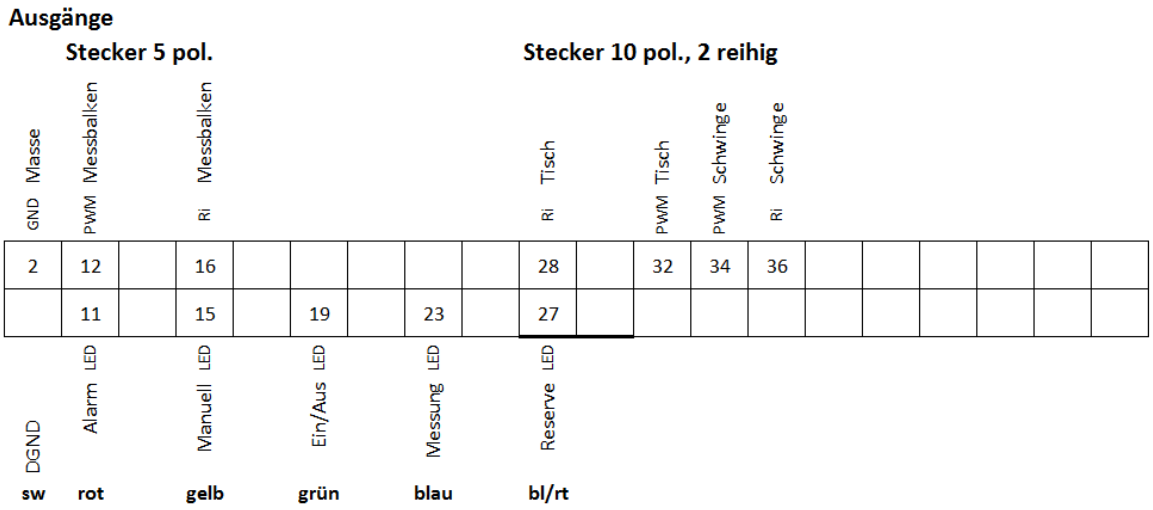


Abbildung 35: Ausgänge – Anschlussbelegung an J2

Als Hilfsmittel für die Programmierung und Tests wurde durch den Autor ein Diagnoseboard, siehe Abbildung 36, entwickelt. Dieses enthält alle Bedienelemente des Prüfstandes sowie Anschlussmöglichkeiten für die Lichtschranken der Positionssysteme. Durch den Anschluss dieses Boards an die MasterCard, an J2 und J3, können vorab alle Funktionen und Reaktionen in Verbindung mit der Programmierung getestet werden, bevor die Antriebs- und Hydraulikkomponenten angeschlossen werden.

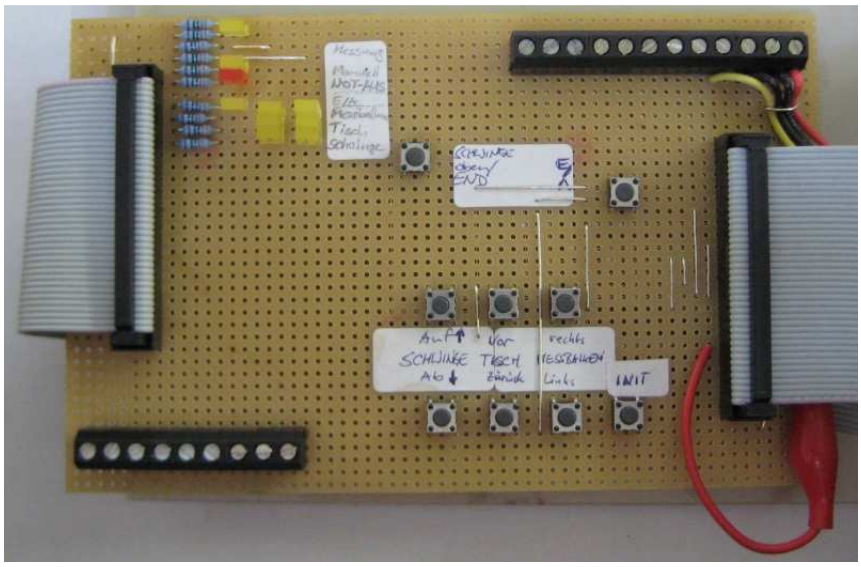


Abbildung 36: Diagnoseboard
Aufbau entspricht dem Bedienermenü, siehe Abbildung 9

5.3.5 Gesamtaufbau der Bedieneinheit

Abbildung 37 und Abbildung 38 zeigen im Vergleich zu Abbildung 6, Abbildung 10 und Abbildung 11 [2] den Aufbau der Bedieneinheit sowie die Unterbringung der elektronischen Baugruppen. Gut zu erkennen sind die Steckverbinder auf der Rückseite der Bedieneinheit. Kabel und Leitungen werden nicht direkt in das Gerätegehäuse geführt, der Anschluss der Stromversorgung entspricht geltenden Standards. Die Änderungen der Bedieneinheit ermöglichen jetzt auch eine dezentrale Aufstellung bspw. auf einem Tisch oder beweglichem Träger.

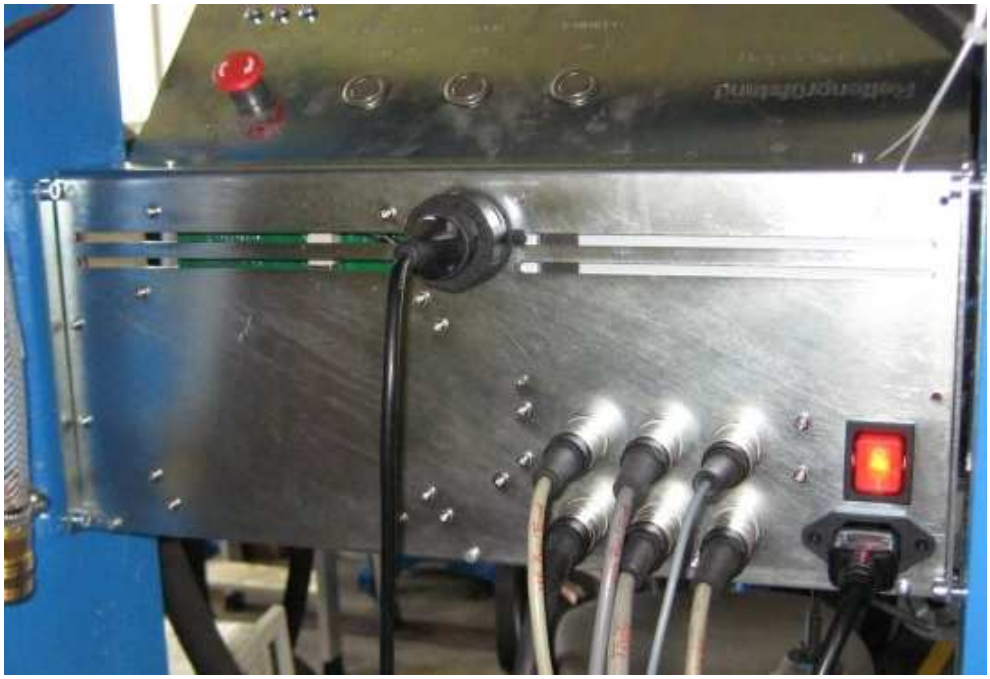


Abbildung 37: Geräterückseite mit Schalter und Steckverbindern

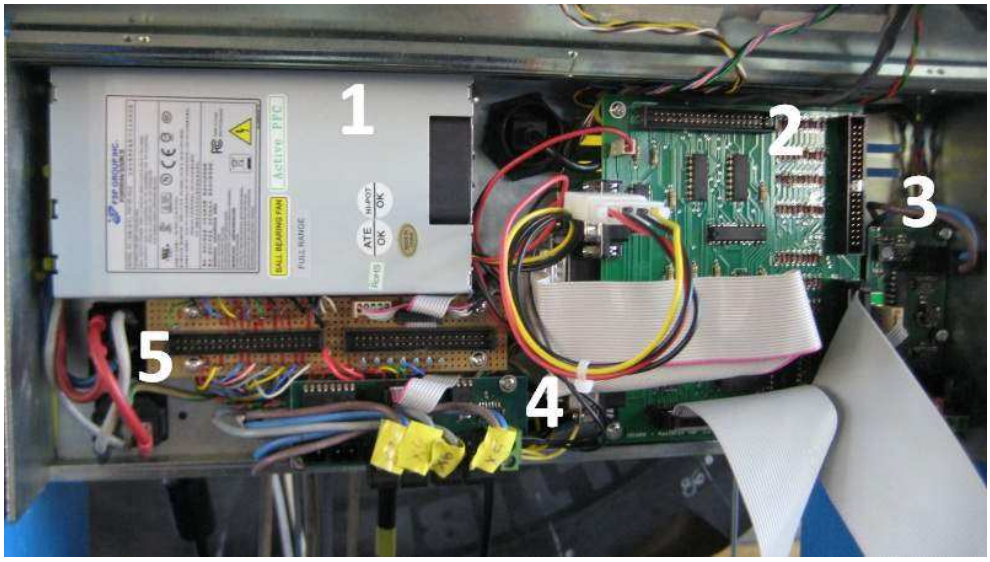


Abbildung 38: Geräteinnenansicht

Abbildung 38: Geräteinnenansicht zeigt die Verteilung der Komponenten im Gerät. Gut zu erkennen sind die einzelnen Baugruppen, wie:

1 – Netzteil

2 – MasterCard [9]

3 – Motorplatine für den Messbalken [2]

4 – Hydraulikplatine für Radlast und Tischbewegung [2]

5 – I/O-Platine für den Anschluss der internen Ein- und Ausgänge an die MasterCard.

Alle Baugruppen sind gut zugänglich, lediglich die USB-ExpasionCard befindet sich unterhalb der MasterCard. Änderungen an dieser Karte sind aber nicht notwendig. Aus diesem Grund stellt das keinen Nachteil dar.

6. Softwareentwicklung

6.1 Steuercontroller

Der derzeit eingesetzte Steuercontroller ist zwar hochflexibel, arbeitet aber an der Leistungsgrenze. In Verbindung mit der Software kommt es vor, dass im Controller Informationen der Inkrementalgeber, der Schlitzscheibe oder der Daten vom DataScan nicht berücksichtigt werden und damit für die Ermittlung der Positionen bzw. Messwerte „verloren“ gehen. Der Controller ist mit der Erfassung der Messwerte sowie der Weitergabe an den PC beschäftigt.

Eine Anpassung der vorhandenen Software stellt einen nicht genau kalkulierbaren zeitlichen Aufwand dar und birgt Gefahren auf die Funktionsfähigkeit des Gesamtsystems. In Verbindung mit der intern festgelegten Struktur des Controllers ist eine komplette Beseitigung der aufgetretenen Probleme nicht zu erwarten. Dies gilt ebenfalls in Bezug auf verschiedene Designfehler bspw. in Bezug auf die Positionierungssysteme für den Messbalken und den Tisch. Aus diesem Grund lohnt es sich über einen Ersatz des Controllers nachzudenken.

Wie bereits unter 5.3 beschrieben, kommt ein HardwareBoard der Firma OpenCockpits [3] zum Einsatz. Der elektrische Aufbau sowie grundsätzliche Parameter wurden dort bereits erläutert.

Die Programmierung erfolgt über das auf IP-basierende IOCP-Protokoll. In Verbindung mit einer USB-ExpansionCard kann der Anschluss an einen PC per USB realisiert werden. Aber auch an einer parallelen Schnittstelle kann eine MasterCard genutzt werden. An eine USB-ExpansionCard können bis zu 4 MasterCards angeschlossen werden. Zusätzlich stehen dort 4 analog-Eingänge, allerdings mit lediglich 8 Bit Auflösung, zur Verfügung. An einer USB-ExpansionCard lassen sich somit maximal 152 digitale Ausgänge, 288 digitale Eingänge sowie 16 DisplayCards mit je 16 digits ansteuern. Für die meisten Anwendungsfälle ist das mehr als ausreichend.

Die Konfiguration der Ein- und Ausgänge erfolgt sehr einfach über eine Konfigurationsdatei, die zusammen mit einer IOCP-Serverkomponente geladen wird.

Eine wesentliche Erweiterung der Funktionalität verbunden mit einer sehr hohen Flexibilität erfolgt auf der Basis eines IOCP-Wrappers [8], der das komplette Kommunikationsprotokoll kapselt. Auf dieser Basis kann die Programmierung auf einfache Weise in C# erfolgen. Für die Nutzung dieser IOCP-Wrapperklasse, von Klaus Aschenbrenner, liegt mir die Genehmigung zur Nut-

zung in eigenen Projekten vor (siehe Anlage 1, Genehmigung zur Verwendung der IOCP-Wrapper-Klasse durch Klaus Aschenbrenner).

Es können sowohl in Hardware vorhandene Taster und Schalter angesprochen als auch Softwareoberflächen entwickelt werden. Bei den Software-Oberflächen werden Tastendrücke in Befehle umgesetzt, die per Ereignis den Druck auf einen Hardwaretaster auslösen.

Die Informationen, an denen der Programmierer interessiert ist, werden im Programm bekannt gemacht und können dann für Aktionen in der Software benutzt werden. Alle Änderungen auf dem IOCP-Bus werden somit überwacht. Die Software ist auf dieser Weise komplett Ereignisorientiert und reagiert auf die gewünschten Statusänderungen.

Bei der Programmierung greift man auf die in der Konfigurationsdatei festgelegten Variablennummern zu. Es empfiehlt sich dort sprechende Bezeichnungen zu vergeben um jederzeit einen Überblick über die Ports und deren Funktionen zu behalten.

Folgendes Zitat untermauert diese Aussage: „Hardware-Boards – Damit Sie überhaupt Hardware, wie Schalter, Displays, LEDs, Potentiometer oder Servomotoren ansteuern und programmieren können, benötigen Sie zunächst fertig bestückte Boards, an die Sie diese Teile anschließen, Die Boards enthalten verschiedene Mikrocontroller, welche die eigentliche Interaktion mit der angeschlossenen Hardware durchführen. Ihre Software muss sich nur um die Kommunikation mit dem Board kümmern – die einzelnen Bauteile werden durch die Boards abstrahiert.

Im Prinzip können Sie diese Abstraktion mit einer mehrschichtigen Geschäftsanwendung vergleichen, wo Sie ebenfalls nur über eine oben liegende Schicht mit der Schicht darunter kommunizieren können. Hier ist die unterste Schicht die Hardware, die Sie programmieren möchten. … Ganz ähnlich abstrahieren die Boards den Zugriff auf die Bauteile. Sie bieten ein API oder ein SDK an, über das Sie die mit dem Board verbundene Hardware programmieren und somit steuern können.“ [7], S. 16

Klaus Aschenbrenner beschreibt die Kommunikation mit der Hardware wie folgt „Die Kommunikation mit den Boards von OpenCockpits erfolgt über ein auf TCP basierendes Netzwerkprotokoll. Das Protokoll namens IOCard Protocol (IOCP) wurde speziell für diese Boards entwickelt. Jedes Mal, wenn ein angeschlossenes Bauteil einen Impuls auslöst (also einen Input-Event gene-

riert), wird eine entsprechende Nachricht über IOCP gesendet, die Sie anschließend in Ihrer Anwendung auswerten und weiterverarbeiten können.

Möchten Sie hingegen Output-Events generieren, etwa um eine LED einschalten oder eine Ziffer anzeigen, so müssen Sie über IOCP eine entsprechende Nachricht senden, die von den Boards empfangen und an die Bauteile weitergeleitet wird.“ [7], S. 20

Bevor auf die Hardware zugegriffen werden kann, müssen die Ein- und Ausgänge festgelegt werden. Dies erfolgt mit der SIOC-Serversoftware.

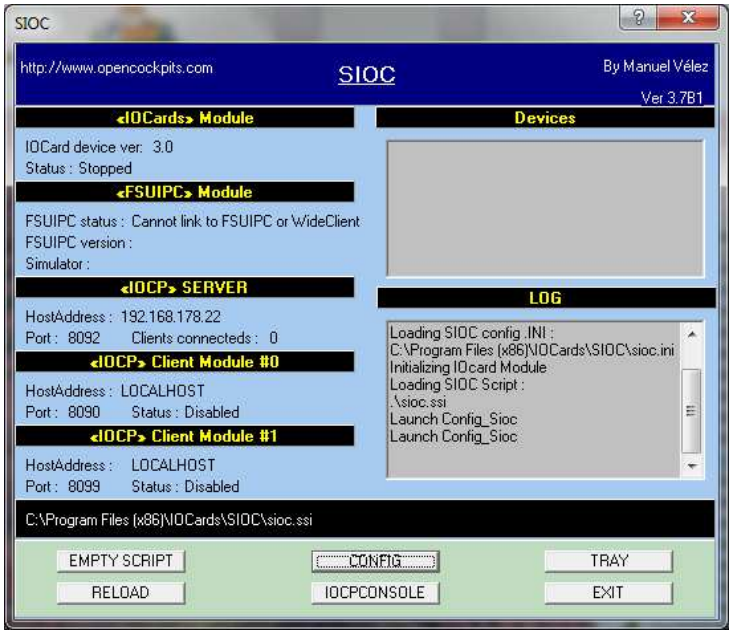


Abbildung 39: SIOC – Serverkomponente

Abbildung 39: SIOC zeigt die Serverkomponente der OpenCockpit-Boards. Mit dieser Software werden die Informationen zwischen den Boards, der Bauteile und der Software ausgetauscht und Konfiguriert. Im Bereich Devices werden angeschlossene Boards angezeigt, aktuell ist kein Board angeschlossen. Für die korrekte Funktion der Konfiguration muss die Device-Nummer, hier für **Device 20**, mit dem folgenden Eintrag der **sioc.ini im selben Ordner der aufzurufenden sioc.exe** übereinstimmen.

```
[MASTERS]
MASTER=0,4,1,20
```

Auf der Abbildung 40, wird die Konfiguration für einen Schalter dargestellt. „Link to“ steht für den Typ des angeschlossenen Bauteils, im Beispiel einen Schalter/Taster. „Var num.“ stellt die Variablennummer bereit, die für die Kommunikation mit der PC-Software benötigt wird. Diese muss eindeutig sein. Es können insgesamt 10.000 Variablen definiert werden, 0 bis 9999. Der „Name“ ist lediglich für eine bessere Lesbarkeit erforderlich, „Description“ steht für einen Kommentar, der ebenfalls die Lesbarkeit der Konfiguration verbessert. Der im Bereich IOCARDS DATA, unter „INs/OUTs/#“ eingetragene Wert entspricht dem eigentlichen Anschluss entsprechend der Anschlussbelegungen in Abbildung 34: Eingänge – Anschlussbelegung an J3 und Abbildung 35: Ausgänge – Anschlussbelegung an J2.

Diese Informationen werden in einem internen Dateiformat, der `sioc.ssi` gespeichert, die beim Programmstart automatisch geladen wird. Die Software SIOC Config bietet Möglichkeiten des Exports und des Imports in Form von Textdateien an, die dann mit jedem Texteditor bearbeitet werden können.

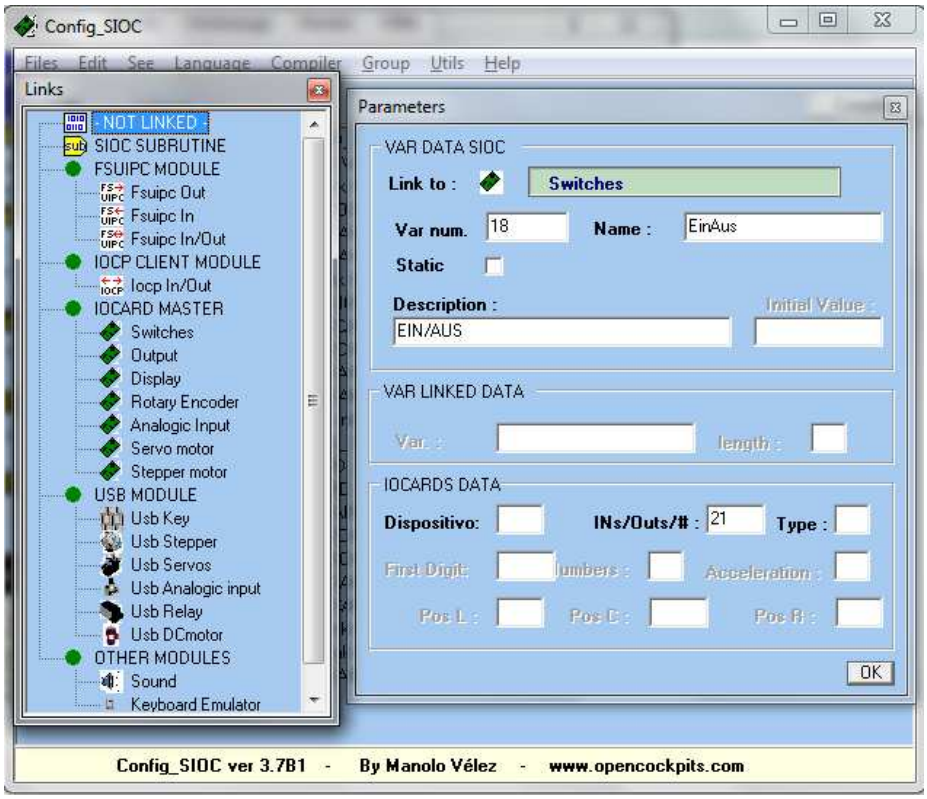


Abbildung 40: SIOC - Config

Mit der IOCPConsole können im laufenden Betrieb die Statusänderungen der einzelnen Ports kontrolliert und manipuliert werden. Auf Abbildung 41: IOCP – Console ist zu erkennen, dass der Wert für die Variable 0 = NotAus manuell auf 1 gesetzt wurde. Durch den gewählten Log On-

Modus wird die Statusänderung „0 = 1 - NotAus“ protokolliert. Mit diesem Mechanismus kann die Konfiguration sowie das Verhalten der gesamten Konfiguration einfach geprüft werden.

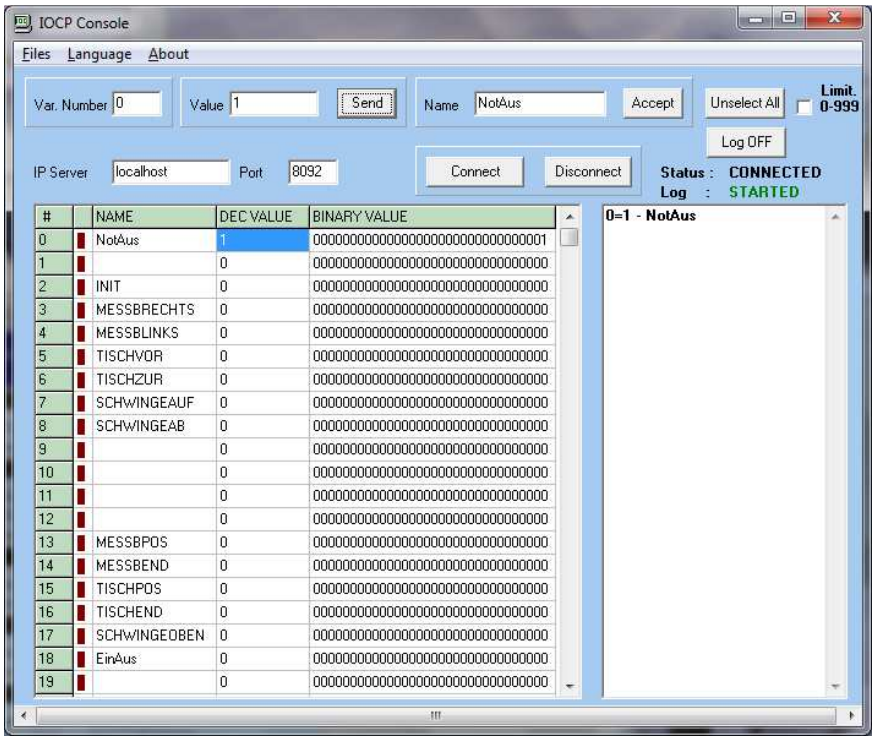


Abbildung 41: IOCP – Console

6.1.1 Installation der SIOC-Software

Die durch OpenCockpits zum Download zur Verfügung gestellte Sioc_37B1.zip enthält als einzige Datei die `setup.exe`. Die Installation erfolgt dabei im Ordner Programme, bei 64-Bit-Systemen unter `C:\Program Files (x86)\IOCards\SIOC`. Da sich auch die Konfigurationsdateien in diesem Ordner befinden, muss der Anwender Schreibrechte besitzen, die standardmäßig in diesem Bereich nicht eingeräumt sind. Eine Installation in dieser Form ist nicht zwingend erforderlich und wird durch den Autor nicht empfohlen. Der Ordner `SIOC` kann ohne Probleme in den Ordner der Anwendung kopiert/verschoben werden. Somit können auch unterschiedliche Versionsstände und Konfigurationen getestet und geprüft werden. Beim Start der `sioc.exe` werden alle Konfigurationsdateien im selben Ordner erwartet.

Beim Aufruf der Serverkomponente unter Windows 7 wird eine Fehlermeldung „Error 1527“ mit Verweis auf die fehlende Datei `IO.SYS` angezeigt. Diese Fehlermeldung beeinflusst die Programmausführung nicht und kann prinzipiell bestätigt werden, ist aber störend. Dieser Treiber wird im Zusammenhang mit dem direkten Anschluss der MasterCard an einen Parallel Port benötigt. In der `sioc.ini` ist der Eintrag im Abschnitt

```
[IOCARDS MODULE]
IOCard_LPT=Yes
```

auf

```
IOCard_LPT=No
```

zu ändern.

Ein Neustart des IOCP-Servers erfolgt ab sofort ohne Fehlermeldung.

6.2 Anwenderprogramm

Die Anwendungslösung wurde in eine einzige Programmierumgebung zusammengeführt. Auf der Basis von C#.Net werden die Steuerungsaufgaben, die Nutzeroberfläche, die Erfassung der Messwerte und die Datenspeicherung realisiert. Die Lösung basiert auf einen vollständig objektorientierten Ansatz. Durch die Bereitstellung der IOCP-Wrapperklasse [7] wird der Zugriff auf die Hardware erheblich vereinfacht. Objektorientierung wird wie folgt beschrieben [13], S. 151 f, Objektorientiertes Programmieren:

„Objektorientierte Programmierung ist ein Denkmuster, bei dem Programme als Menge von über Nachrichten kooperierenden Objekten organisiert werden und jedes Objekt Instanz einer Klasse ist. … Die OOP verlangt eine Anpassung des Software-Entwicklungsprozesses und der eingesetzten Methoden an den Denkstil des Programmierers – nicht umgekehrt! …

Der Programmierer versteht unter einem Objekt die Zusammenfassung (Kapselung) von Daten und zugehörigen Funktionalitäten. … Objekte sind ganz allgemein Dinge, die Sie in Ihrem Code beschreiben wollen, es sind Gruppen von Eigenschaften, Methoden und Ereignissen, die logisch zusammengehören. Als Programmierer arbeiten Sie mit einem Objekt, indem Sie dessen Eigenschaften und Methoden manipulieren und auf seine Ereignisse reagieren.

… Eine Klasse ist nicht mehr und nicht weniger als ein „Bauplan“, auf dessen Grundlage die entsprechenden Objekte zur Programmlaufzeit erzeugt werden. Gewissermaßen als Vorlage (Prägestempel) für das Objekt legt die Klasse fest, wie das Objekt auszusehen hat und wie es sich verhalten soll. Es handelt sich bei einer Klasse also um eine reine Softwarekonstruktion, die Eigenschaften, Methoden und Ereignisse eines Objektes definiert, ohne das Objekt zu erzeugen.“

Der Lebenszyklus eines Objektes kann wie folgt beschrieben werden:

„…

- Referenzieren (eine Objektvariable wird deklariert, sie verweist momentan noch auf null)
- Instanziierung (die Objektvariable zeigt jetzt auf einen konkreten Speicherplatzbereich)
- Initialisierung (die Datenfelder der Objektvariablen werden mit Anfangswerten gefüllt)
- Arbeiten mit dem Objekt (es wird auf Eigenschaften und Methoden des Objekts zugegriffen, Ereignisse werden ausgelöst)

- Zerstören des Objektes (das Objekt wird dereferenziert, der belegte Speicherplatz wird wieder freigegeben)“ [13], S. 156, Das Erzeugen eines Objektes

Static-Variablen – Anwendungweit

Im Zusammenhang mit der Objektorientierten Programmierung werden demzufolge i. d. R. Variablen immer als Instanz betrachtet. Man nutzt also eine Kopie des Inhaltes für weitere Auswertungen. In der Anwendung ist es aber häufig notwendig nicht nur auf Kopien der Inhalte zuzugreifen sondern direkt auf deren Inhalt. Für diesen Fall können die betreffenden Variablen public static vereinbart werden. Die Inhalte dieser Variablen wurden für die konkrete Anwendung in zwei Property-Klassen definiert. Dies betrifft die Klasse CPosition, dort werden alle Positionsdaten, die Messbalkenposition und die Tischposition betreffend, abgespeichert. Die zweite Klasse CZustand enthält alle Zustandsdaten der Steuerung und alle für die Messwerterfassung notwendigen Variablen.

Für den Zugriff auf diese Daten muss keine Instanz erzeugt werden, vielmehr erfolgt der Zugriff Anwendungsweit durch Voranstellen des Klassenbezeichners. Bspw. CPosition.PosMessbalkenIst liefert die Ist-Position des Messbalkens, CZustand.MESSBALKENdir gibt Auskunft über die Bewegungsrichtung des Messbalkens.

Im Punkt 6.1 Steuercontroller wurde schon kurz auf die Möglichkeiten der HardwareBoards von OpenCockpits [3] eingegangen. In Bezug auf die Konfiguration der Bauteile wurde bereits Bezug zum IOCP-Kommunikationsprotokoll genommen.

Für die Programmierung wird im Folgenden die Kommunikation genauer betrachtet und der interne Mechanismus erläutert. „Damit Sie über Änderungen an IOCP-Variablen informiert werden, müssen Sie beim Start Ihrer Software bekannt geben, welche IOCP-Variablen dies betrifft. Interessieren Sie sich zum Beispiel für Veränderungen der IOCP-Variablen 4, 5, 6 und 7, so müssen Sie den folgenden String über den TCP-Port an den IOCP-Server senden: Arn.Inicio:4:5:6:7 … Diesen String empfängt der IOCP-Server und merkt sich intern, an welchen Variablen Sie interessiert sind. Sie, bzw. Ihr Programm wird somit nur über Veränderungen informiert, die Sie ausdrücklich sehen möchten. … Wenn sich nun eine der IOCP-Variablen verändert hat, an denen Sie Interesse bekundet haben, sendet Ihnen der IOCP-Server den folgenden String: Arn.Resp:4=1:5=0:6=55

Wie Sie sehen, erhalten Sie für jede veränderte Variable deren aktuellen Wert. Im Beispiel haben sich gleichzeitig drei Variablen geändert:

Die IOCP-Variable 4 hat den Wert 1, die Variable 5 hat den Wert 0 und die Variable 6 hat den Wert 55. Da sich der Wert der Variablen 7 nicht geändert hat wurde kein Wert gesendet.

Hätten sich die drei Variablen zu unterschiedlichen Zeitpunkten geändert würde der IOCP-Server diese Änderungen einzeln senden, etwa so:

Arn.Resp:4=1
Arn.Resp:5=0
Arn.Resp:6=55

Um Änderungen der Werte von Variablen über das IOCP-Protokoll zu versenden, genügt es, den neuen Wert der Variablen wieder per String an den IOCP-Server zu senden, beispielsweise so:

Arn.Resp:6=35
Arn.Resp:7=118

In diesem Beispiel haben Sie nun den Wert der Variablen 6 und 7 geändert. Die Variable 6 könnte zum Beispiel ein Potentiometer darstellen und Variable 7 könnte eine direkte Verbindung zu einer der 7-Segmentanzeigen repräsentieren, die an eine DisplayCard angeschlossen sind.
Wenn nun andere IOCP-Clients an den Variablen 6 und 7 interessiert sind, werden diese wiederum die Variablenänderung über das IOCP-Protokoll zugestellt bekommen. … Um Ihnen die Arbeit mit dem IOCP-Protokoll zu erleichtern möchte ich nun ein Framework vorstellen, das die komplette IOCP-Interaktion abstrahiert.

Dieses Framework bietet die folgende Abstraktions-Ebene an:

- die Kommunikation zwischen Ihrer Software und dem IOCP-Server wird über eine Framework-Klasse implementiert
- ändert sich eine IOCP-Variable, wird eine von Ihnen definierte Methode im .Net-Code aufgerufen
- IOCP-Variablen können durch einen simplen Methodenaufruf verändert werden

Die komplette Kommunikation auf der Basis des IOCP-Protokolls ist in der Klasse IOCPClient implementiert.“ [7], S. 21 f

Die wichtigsten Methoden der Klasse :

Methode	Beschreibung
Connect	stellt eine Verbindung zu einem IOCP-Server her
RunEventDispatcherAsync	Startet einen Worker-Thread, der sämtliche Änderungen in IOCP-Variablen entgegennimmt und an die entsprechenden Variablen verteilt
SendData	Über diese überladene Methode haben Sie die Möglichkeit, einer IOCP-Variablen einen neuen Wert zuzuweisen
AddVariableChangeListener	Registriert eine neue Klasse (die die eigentliche Hardwaresteuerung implementiert) innerhalb des IOCP-Clients. Ändert sich eine registrierte IOCP-Variable, werden die registrierten Methoden in dieser Klasse aufgerufen

Tabelle 8: Methoden der IOCPClient-Klasse [7]

6.2.1 Festlegung der Anschlüsse

Abbildung 34: Eingänge – Anschlussbelegung an J3 und Abbildung 35: Ausgänge – Anschlussbelegung an J2 zeigen die Anschlussbelegungen für die Ein- und Ausgänge. Diese Konfiguration wird mit den Variablennummern der SIOC-Konfiguration verknüpft, wie folgende Abbildung zeigt.

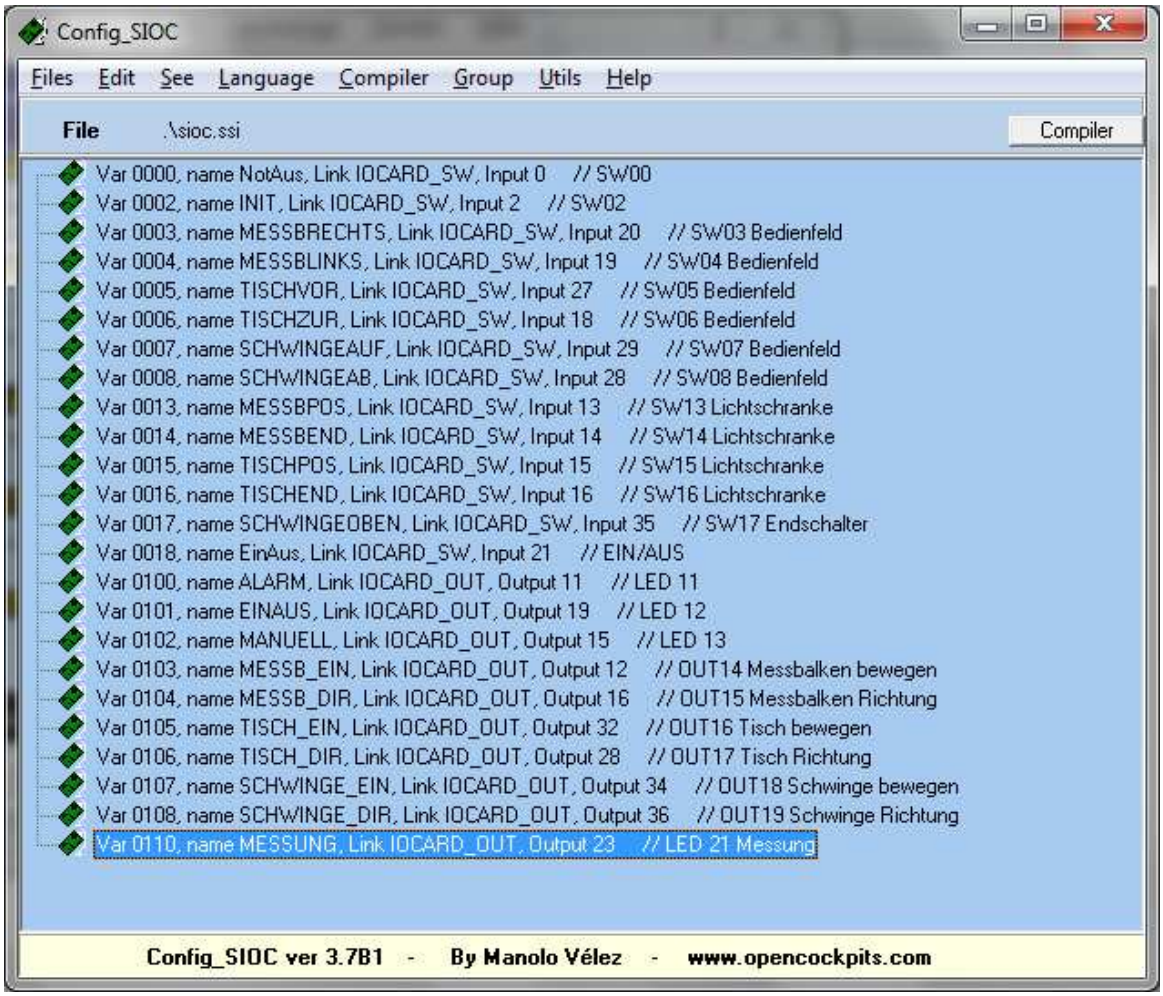


Abbildung 42: SIOC - Definition der Ein-/Ausgabeports

In der Datei EnumReifenPruefstand.cs werden die systemweit gültigen Festlegungen für die Ein- und Ausgabegeräte getroffen. Diese müssen mit den Festlegungen der Variablennummern der SIOC-Definitionen, wie in Abbildung 42 dargestellt, übereinstimmen.

```
public enum Inputs : int
{
    NotAus = 0,             // Notaus
    Init = 2,               // INIT-Taster
    MessBalkenRechts = 3,   // Taster Bedienfeld
    MessBalkenLinks = 4,    // Taster Bedienfeld
    TischVor = 5,           // Taster Bedienfeld Tisch vorwärts
    TischZur = 6,           // Taster Bedienfeld Tisch rückwärts
    SchwingeAuf = 7,        // Taster Bedienfeld Schwinge hoch
    SchwingeAb = 8,         // Taster Bedienfeld Schwinge runter
    // kein echter Anschluss: Schwinge belasten solange AuflastIst <= AuflastSoll
    SchwingeLast = 9,
    MessbalkenPos = 13,     // Lichtschranke Messbalken Position
    MessbalkenEnd = 14,     // Lichtschranke Messbalken Endpositionen
    TischPos = 15,          // Lichtschranke Tischposition
    TischEnde = 16,         // Lichtschranke Endpositonen/Mitte
    SchwingeEnd = 17,       // Endschalter Rad/Schwinge oben
    EinAus = 18,            // Ein/Aus
    // kein echter Anschluss: Tisch vor, bis CZustand.TischPos == TischPosSoll
    TischPosAutoMod = 20,
    // kein echter Anschluss: Messbalken rechts, bis CZustand.MBPos == MBPosSoll
    MBPosAutoMod = 21
}

public enum Outputs : int
{
    LEDALARM = 100,             // LED Bedienfeld opt. Erroranzeige
    LEDEIN = 101,               // LED Bedienfeld Arbeitsbereitschaft
    MESSBALKENEIN = 103,        // 0 - aus, 1 - ein
    MESSBALKENRICHTUNG = 104,   // 0 nach rechts, 1 nach links
    TISCHEIN = 105,             // 0 - aus, 1 - ein
    TISCHRICHTUNG = 106,        // 0 - zurück, 1 - vorwärts
    SCHWINGEEIN = 107,          // 0 aus, 1 - ein
    SCHWINGERICHTUNG = 108,     // 0 nach unten, 1 nach oben
    LEDMESSUNG = 110            // LED Bedienfeld - Status Messung
}
```

Der Zugriff im Programm erfolgt bspw. für Eingänge in dieser Form (int)Inputs.EinAus bzw. für Ausgänge in dieser Form (int)Outputs.MESSBALKENRICHTUNG. Mit (int) werden die Wertzuweisungen als Integer geparst da diese ansonsten als Objektvariablen betrachtet werden und somit nicht zugewiesen werden können.

Dieses Vorgehen vereinfacht Änderungen, die möglicherweise bei der Beschaltung der Hardware notwendig sind, erheblich. Angenommen der Ein-/Aus-Taster soll anstatt am Eingabeport 21 am Port 26 angeschlossen werden, so ist lediglich in der SIOC-Konfiguration für die Variablennummer 0018 unter „INs/OUTs/#“ der Wert auf 26 zu setzen und zu speichern. Durch den Neustart der SIOC-Serverkomponente sind alle Änderungen vollzogen. Da der Zugriff des Anwendungsprogrammes auf die Variablennummer der SIOC-Server-Konfiguration erfolgt, entsteht eine Entkopplung gegenüber der Hardware, die notwendig werdende Änderungen im Layout sehr erleichtern.

6.2.2 Die Klasse Steuerung

Nebenstehende Abbildung zeigt die Klassenstruktur für die Hardwaresteuerung des Reifenprüfstandes.

Diese Klasse erbt von der bereitgestellten Klasse IOCPBase [7]. Dadurch wird die Implementierung der Aktionen, die beim Eintreten von IOCP-Variablen-Änderungen ausgeführt werden deutlich vereinfacht.

Beispielhaft werden nachfolgend die typischen Methoden für die Klasse Steuerung.cs erläutert.

- OnEinAusEvent
- OnMessBalkenLinksEvent
- OnMessbalkenPosEvent
- OnMessbalkenEndEvent
- OnSchwingeLastEvent

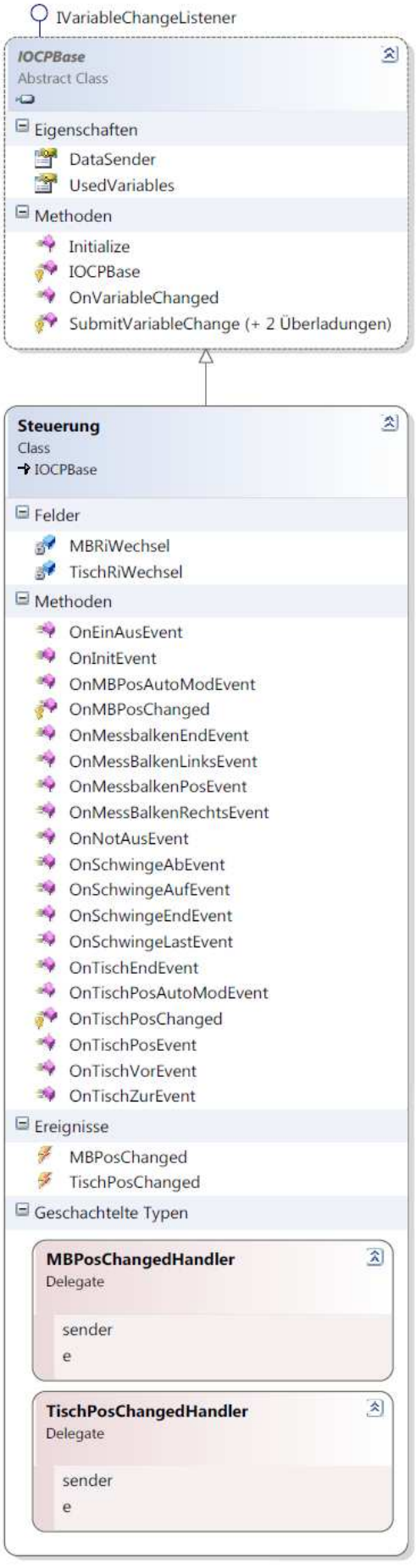


Abbildung 43: Klassendiagramm Steuerung

6.2.3 Reaktion auf Tastendruck – Ein/Aus mit Speicherung des Zustandes

```
/// <summary>
/// Diese Methode wird ausgeführt, wenn der Ein-Taster betätigt wird.
/// Der Zustand wird in _EinAus gespeichert.
/// 0 - Aus, 1 - Ein
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
[Binding(EventID = (int)Inputs.EinAus)]
public void OnEinAusEvent(object sender, InputSwitchEventArgs e)
{
    if (e.KeyState == KeyState.Pressed)
    {
        if (CZustand.NotAus == false)
        {
            CZustand.EinAus = !CZustand.EinAus;

            if (CZustand.EinAus == true)
            {
                SubmitVariableChange((int)Outputs.LEDEIN, 1);
                //Console.WriteLine("Ein");
            }
            else
            {
                SubmitVariableChange((int)Outputs.SCHWINGEEIN, 0);
                SubmitVariableChange((int)Outputs.TISCHEIN, 0);
                SubmitVariableChange((int)Outputs.MESSBALKENEIN, 0);
                SubmitVariableChange((int)Outputs.SCHWINGERICHTUNG, 0);
                SubmitVariableChange((int)Outputs.TISCHRICHTUNG, 0);
                SubmitVariableChange((int)Outputs.MESSBALKENRICHTUNG, 0);
                SubmitVariableChange((int)Outputs.LEDEIN, 0);
                //Console.WriteLine("Aus");
            }
        }
    }
}
```

Die Methode für das Einschalten des Reifenprüfstandes speichert den Zustand in der Zustandsvariablen CZustand.EinAus. Alle weiteren Aktionen sind von diesem Zustand abhängig, d. h. die Aktionen können nur ausgeführt werden wenn der Reifenprüfstand auch eingeschaltet ist. Ein Statuswechsel ist aber nur möglich, wenn NotAus nicht ausgelöst ist. Die im Einschalter eingebaute LED wird eingeschaltet und signalisiert diesen Status. Wird der Status zurückgesetzt (false), werden alle Antriebe ausgeschaltet.

6.2.4 Reaktion auf Tastendruck – Messbalken nach links

```
/// <summary>
/// Diese Methode wird ausgeführt, bei Taster Messbalken links.
/// Ausgabe MessBalkenLinks, Richtung = 1
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
[Binding(EventID = (int)Inputs.MessBalkenLinks)]
public void OnMessBalkenLinksEvent(object sender, InputSwitchEventArgs e)
{
    if ((CZustand.NotAus == false) && (CZustand.EinAus == true)
                                   && (CZustand.MESSBAKENendLi == false))
    {
        if ((e.KeyState == KeyState.Pressed))
        {
            Console.WriteLine("Messbalken links");
            if ((CZustand.MESSBALKENdir == true)
                                        && (!CZustand.MESSBALKENendRe))
                  MBRiWechsel = true;
            CZustand.MESSBALKENdir = false;
            SubmitVariableChange((int)Outputs.MESSBALKENRICHTUNG, 1);
            SubmitVariableChange((int)Outputs.MESSBALKENEIN, 1);
            CZustand.MESSBALKENendRe = false;
        }
        else
        {
            SubmitVariableChange((int)Outputs.MESSBALKENEIN, 0);
        }
    }
}
```

Oben abgebildete Methode zeigt den typischen Aufbau für die Reaktion auf Events, die durch Tastendruck ausgelöst werden und speziell für MessBalkenLinks. Dieses wird durch Druck auf den zugehörigen Taster an der Bedieneinheit oder durch Betätigen des Tasters der Nutzeroberfläche ausgelöst. Die eigentliche Aktion wird mit SubmitVariableChange eingeleitet. Dies aber nur, wenn die Eingangsbedingungen erfüllt sind. D. h., NotAus ist nicht betätigt, der Reifenprüfstand ist eingeschaltet und der Messbalken sich nicht in der linken Endposition befindet. Wenn der Tasters gedrückt wurde und damit KeyState.Pressed, wird als nächstes geprüft ob sich die Bewegungsrichtung geändert hat. Dieser Status ist für die korrekte Berechnung der Positionen notwendig. Die Richtung des Messbalkens wird in der Zustandsvariablen MESSBALKENdir auf false gesetzt. Die Steuerelektronik für den Messbalkenantrieb benötigt für die `MESSBALKENRICHTUNG` = 1, mit MESSBALKENEIN = 1 wird der Motor eingeschaltet.

Beim loslassen des Tasters MessbalkeLinks wird das Event erneut ausgelöst, aber im Programmablauf der alternative Zweig durchlaufen. Dort wird der Antriebsmotor mit MESSBALKENEIN = 0 ausgeschaltet.

6.2.5 Positionsberechnung Messbalken-/Tischposition

```
/// <summary>
/// Messbalken Position = MessbalkenPos
/// 1 = Lichtschranke offen
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
[Binding(EventID = (int)Inputs.MessbalkenPos)]
public void OnMessbalkenPosEvent(object sender, InputSwitchEventArgs e)
{
    if ((CZustand.NotAus == false) && (CZustand.EinAus == true))
    {
        if ((e.KeyState == KeyState.Released) && (CZustand.MESSBALKENdir
                                                            == true))
        {
            Console.WriteLine("Messbalken rechts Pos+");
            if (CZustand.MESSBALKENendRe == true)
            {
                CZustand.MESSBALKENendRe = false;
            }
            else
            {
                if (!MBRiWechsel)
                    CPosition.PosMessbalkenIst++;
                else
                    MBRiWechsel = false;
                this.OnMBPosChanged();
            }
        }

        if ((e.KeyState == KeyState.Pressed)
                                && (CZustand.MESSBALKENdir == false))
                                            // Messbalken nach links
        {
            Console.WriteLine("Messbalken links Pos-");
            if (CZustand.MESSBAKENendLi == true)
            {
                CZustand.MESSBAKENendLi = false;
            }
            else
            {
                if (!MBRiWechsel)
                    CPosition.PosMessbalkenIst--;
                else
                    MBRiWechsel = false;
                this.OnMBPosChanged();
            }
        }
    }
}
```

Die Methode für die Ermittlung der eingenommenen Position prüft als Eingangsbedingung, dass NotAus nicht betätigt wurde und der Prüfstand eingeschaltet ist. In Abhängigkeit des Zustands-wechsels KeyState.Released = Lichtschranke wechselt von ein nach aus bzw. KeyState.Pressed = Lichtschranke wechselt von aus nach ein sowie der Bewegungsrichtung, wird entweder um eine

Position erhöht bzw. um eine Position verringert. Bei einem Richtungswechsel darf die Position erst beim zweiten Auftreten des Ereignisses erhöht bzw. verringert werden. Dies ist notwendig, da der Status der Lichtschranke beim Richtungswechsel ebenfalls sofort wechselt, der Messbalken sich aber minimal bewegt hat. In diesem Fall wird lediglich die Zustandsvariable für den Richtungswechsel zurückgesetzt. Des Weiteren wird in Abhängigkeit der Richtung der jeweilige Endlagenzustand rechts oder links zurückgesetzt.

Nachdem sich die Position geändert hat wird das Event OnMBPosChanged gefeuert. Das ist notwendig um die neue Position in der Benutzeroberfläche zur Anzeige zu bringen.

6.2.6 Bestimmung der Endposition Messbalken/Tisch

```
/// <summary>
/// Messbalken Ende = MessbalkenEnd
/// 1 = Lichtschranke offen
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
[Binding(EventID = (int)Inputs.MessbalkenEnd)]
public void OnMessbalkenEndEvent(object sender, InputSwitchEventArgs e)
{
    if ((CZustand.NotAus == false) && (CZustand.EinAus == true))
    {
        if (e.KeyState == KeyState.Pressed)
        {
            Console.Write("Messbalken Ende ");
            SubmitVariableChange((int)Outputs.MESSBALKENEIN, 0);

            if (CZustand.MESSBALKENdir == true)     // Messbalken rechts
            {
                Console.WriteLine("- rechts");
                CZustand.MESSBALKENendRe = true;
                CPosition.PosMessbalkenIst = CZustand.MBPosMax + 1;
            }
            else
            {
                Console.WriteLine("- links");
                CZustand.MESSBAKENendLi = true;
                CPosition.PosMessbalkenIst = 0;
            }
        }
    }
}
```

Die Methode ermittelt die Endposition und prüft als Eingangsbedingung, dass NotAus nicht betätigt wurde und der Prüfstand eingeschaltet ist. Tritt dieses Ereignis auf, wird der Antriebsmotor ausgeschaltet. In Abhängigkeit der Richtung, die beim Erreichen der Endposition gesetzt war wird der Anfangs- bzw. Endwert neu zugewiesen. Damit kann das Positioniersystem automatisch

in einen definierten Ausgangszustand gebracht werden, wenn dies notwendig ist. Die aktuelle Position wird beim normalen Beenden des Programmes gespeichert und beim Starten wiederhergestellt.

Die Methoden für die Messbalken- und Tischposition bzw. die Endpositionen sind vom methodischen Ansatz identisch.

6.2.7 Soll-Ist-Vergleich der Radlast

```
/// <summary>
/// Diese Methode wird aus dem Formular OriginalUI aufgerufen.
/// Soll-Ist-Vergleich
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
//private void SchwingeLast()
[Binding(EventID = (int)Inputs.SchwingeLast)]
public void OnSchwingeLastEvent(object sender, InputSwitchEventArgs e)
{
    if ((CZustand.NotAus == false) && (CZustand.EinAus == true))
    {
        if ((e.KeyState == KeyState.Pressed) ||
            (e.KeyState == KeyState.Released) && (CZustand.SchwingeEin))
        {
            //Console.WriteLine("Schwinge Last");
            double _auflastsoll = CZustand.AuflastSoll * 0.995; // -0,5 %

            if ((CZustand.AuflastIst > 0)
                            && (CZustand.AuflastIst >= _auflastsoll))
                SubmitVariableChange((int)Outputs.SCHWINGEEIN, 0);
            else
            {
                SubmitVariableChange((int)Outputs.SCHWINGEEIN, 1);
            }
        }
    }
}
```

Die Methode OnSchwingeLastEvent führt einen Vergleich zwischen dem vorgegebenen Sollwert für die Radbelastung und dem gemessenem Istwert aus. Der Sollwert wird in der Benutzeroberfläche eingestellt und an die Zustandsvariable CZustand.AuflastSoll übergeben. Der Istwert wird in der Klasse DataScan und dort in der Methode getDataScanData in der Zustandsvariablen CZustand.AuflastIst gespeichert. Wurde der Sollwert erreicht, wird die Hydraulik für die Radbelastung ausgeschaltet. Das erneute Einschalten der Radlast erfolgt, sobald der Sollwert mit 0,5% unterschritten wurde. Die Lasterhöhung beim Einschalten der Hydraulik in der Abwärtsbewegung erfolgt sehr schnell. Die eingesetzten Hydraulikventile kennen aber nur die Zustände Ein oder Aus. Dadurch ist die Belastung kurzfristig höher, als der gewählte Sollwert. Durch den ge-

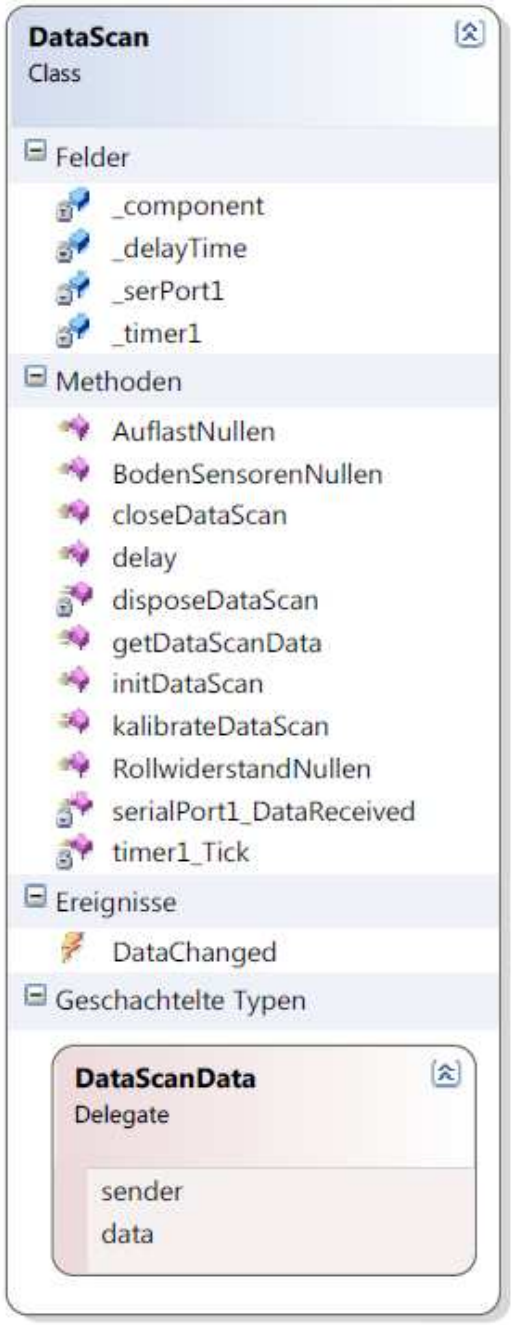
wählten Korrekturwert wird die Belastung als Mittelwert des Sollwertes eingenommen. Dazu muss der Volumenstrom der Hydraulik, an der Drossel, so eingestellt werden, dass der Sollwert nur geringfügig überschritten wird.

Diese Methode wird in der Nutzeroberfläche ausgelöst, wenn das Rad durch den Druck auf die Taste „Reifen runter“ belastet wird. Gleichzeitig wird die Zustandsvariablen CZustand.SchwingeEin gesetzt. Dadurch wird mit jedem Eintreffen von Messdaten der Vergleich zwischen dem Soll- und Istwert ausgeführt.

Ist auch nur eine der Eintrittsbedingungen zu Beginn des Methodenaufrufes nicht erfüllt, wie NotAus wurde betätigt oder der Reifenprüfstand ist ausgeschaltet oder der Zustand SchwingeEin ist zurückgesetzt, werden keine Aktionen ausgeführt.

6.2.8 Klasse DataScan, Event – Messwerte

Die Erfassung der Messwerte erfolgt durch die Sensoren, die am DataScan zusammengefasst und über die serielle Schnittstelle bereitgestellt werden.



Für diesen Zweck wurde eine Klasse DataScan entwickelt, die folgende öffentlichen Methoden bereitstellt:

- initDataScan – stellt die Verbindung der seriellen Schnittstelle entsprechend der Einstellungen der Anwendungsoberfläche her
- getDataScanData – initiiert die Messvorgänge in einem definierbaren Intervall oder beendet diesen Vorgang
- DataChanged – feuert ein Ereignis, sobald neue Messwerte vorliegen, diese werden in der Nutzeroberfläche visualisiert, data übergibt den aktuellen Wert für die Radlast
- AuflastNullen, BodenSensorenNullen, RollwiderstandNullen setzt die Anzeigewerte auf null und registriert die Offsetwerte für eine automatische Wertkorrektur
- kalibrateDataScan – Initialisiert DataScan bspw. nach einem Werksreset
- closeDataScan – schließt die serielle Schnittstelle und die Verbindung zum DataScan
- delay – stellt eine Verzögerung zur Verfügung, die zwischen den Initialisierungsanweisungen notwendig ist, die Verzögerungszeit ist als Zahl vom Typ long zu übergeben und stellt eine Wartezeit in ms ein
- die interne Methode serialPort1_DataReceived liest die Daten der seriellen Schnittstelle ein und verteilt diese auf die Variablen für die einzelnen Messwerte

6.2.9 Methode initDataScan

```
public void initDataScan()
{
    CZustand.SerPortBezeichner = Properties.Settings.Default.SerPortName;
    _component = new System.ComponentModel.Container();
    _serPort1 = new SerialPort(this._component);
    _timer1 = new System.Windows.Forms.Timer(this._component);

    // timer1
    _timer1.Interval = 250;   // 100 == 10 Messwerte pro sek, 200 == 5
    _timer1.Tick += new System.EventHandler(this.timer1_Tick);

    // serialPort1
    _serPort1.PortName = CZustand.SerPortBezeichner;   //"COM2";
    _serPort1.BaudRate = 38400;
    _serPort1.ReceivedBytesThreshold = 180;
    _serPort1.DataReceived
         += new SerialDataReceivedEventHandler(serialPort1_DataReceived);
    try
    {
        _serPort1.Open();
        _serPort1.WriteLine("tc\r");          // take controller status
        delay(_delayTime);
        _serPort1.WriteLine("tt01000\r");
        delay(_delayTime);
        // Datenformat auf ASCII setzen
        _serPort1.WriteLine("rm0\r");         // Textmode 0 - large
        delay(_delayTime);

        // WriteLine unter MS sendet nur ein NewLine \n!! deshalb \r
        // string text = serialPort1.ReadExisting();
    }
    catch (Exception ex)
    {
        System.Windows.Forms.MessageBox.Show(ex.Message,
            _serPort1.PortName + ", " + _serPort1.BaudRate + " baud");
    }
}
```

Die Methode initialisiert die serielle Verbindung mit DataScan 7010. Die Datenrate ist mit 38.400 baud festgelegt, aus diesem Grund wird dieser Wert fest vorgegeben. Der Wert ReceivedBytesThreshold = 180 legt fest, dass erst nachdem 180 Byte im Datenpuffer der seriellen Schnittstelle angekommen sind, das Event DataReceived ausgelöst (gefeuert) wird, siehe Methode serialPort1_DataReceived. Alle weiteren Parameter für die Datenverbindung entsprechen dem Standard. Aus diesem Grund erfolgen dafür keine Zuweisungen. Im Ergebnis des Ereignisses DataReceived werden die Daten aus dem Datenpuffer an die Variablen data1 … data18 zugewiesen. Diese Werte werden in Zahlen vom Typ double konvertiert und stehen dann, Anwendungsweit, als Kräfte der Maßeinheit Newton und für die Radlast in Kilonewton, über die Variablen CZustand.Kraft1 … CZustand.Kraft15 (Messwerte der Minikraftsensoren im Messbalken), CZu-

stand.Auflast (Belastung des Rades) und CZustand.Scherkraft (Rollwiderstand) zur weiteren Verarbeitung zur Verfügung.

Zusätzlich wird ein Timer initialisiert, aber noch nicht gestartet, der die Messungen initiiert.

6.2.10 Auslösen eines Messvorganges

```
public void getDataScanData(bool state)
{
    if (state)
    {
        _timer1.Enabled = true;
    }
    else
    {
        _timer1.Enabled = false;
    }
}

private void timer1_Tick(object sender, EventArgs e)
{
    try
    {
        _serPort1.Write("rl1,18\r\n");
    }
    catch
    {
    }
}
```

Mit der Methode getDataScanData wird ein kontinuierlicher Messvorgang eingeleitet, wenn der Status gesetzt (state == true) ist. Dabei wird lediglich der _timer1 gestartet. Der Timer löst ab diesem Moment, entsprechend des eingestellten Intervalls, das Ereignis timer1_Tick aus. Im Ereignishandler wird die Anweisung für den Messvorgang „rl1,18“ an DataScan gesendet. DataScan gibt die aktuellen Sensorwerte über die serielle Schnittstelle aus. Dort wird wiederum das Ereignis DataReceived ausgelöst, welches den Empfang der Daten signalisiert und die ermittelten Daten im System bereitstellt, siehe Methode serialPort1_DataReceived. Wird die Methode getDataScanData mit zurückgesetztem Status (state == false) ausgeführt wird der Timer und somit das Auslesen der aktuellen Sensordaten beendet.

Die auszugsweise dargestellte Methode serialPort1_DataReceived zeigt die Behandlung der vom DataScan an die serielle Schnittstelle gelieferten Daten. Beim Einlesen der Daten sind im Praxisbetrieb mehrfach Fehler aufgetreten. Diese werden im try – catch Konstrukt abgefangen. Für diesen Fall wird der Messvorgang kurzzeitig beendet, die Daten im Datenpuffer verworfen und anschließend wieder neu gestartet.

```
void serialPort1_DataReceived(object sender, SerialDataReceivedEventArgs e)
{
    if (_serPort1.IsOpen)
    {
        int l = _serPort1.BytesToRead;
        string data16 = l.ToString();

        try
        {
            byte[] ba = new byte[l];
            _serPort1.Read(ba, 0, l);   // erstes Byte entfernen

            string data = System.Text.Encoding.Default.GetString(ba, 0, l);
            if (l == 180)
            {
              string data1 = System.Text.Encoding.Default.GetString(ba, 0, 9);
              :
              :
              string data18 = System.Text.Encoding.Default.GetString(ba, 170, 9);

              // Konvertierung der Werte in double
              CZustand.Kraft1 = (Convert.ToDouble(data1) / 100) -
                                                          CZustand.Kraft1Offset;
              :
              :
              CZustand.Kraft15 = (Convert.ToDouble(data15) / 100) -
                                                         CZustand.Kraft15Offset;

              CZustand.Auflast = (Convert.ToDouble(data17) / 100 / 1000) -
                                              CZustand.AuflastOffset;  // in KN
              CZustand.Scherkraft = (Convert.ToDouble(data18) / 100) -
                                            CZustand.ScherkraftOffset; // in N
          }
        }
        catch
        {
            getDataScanData(false);       // Datenerfassung aus
            delay(100);                   // 100 ms warten
            _serPort1.DiscardInBuffer();  // Datenpuffer leeren
            getDataScanData(true);        // Datenerfassung ein
        }
    }
}
```

6.2.11 Auflast, Bodensensoren und Scherkraftsensor nullen

```
public string AuflastNullen()
{
    string s = String.Empty;

    getDataScanData(false);
    CZustand.AuflastOffset += CZustand.AuflastIst;
    getDataScanData(true);

    s = "Auflast genullt";
    return (s);
}
```

Für die Ermittlung der korrekten Messwerte ist es erforderlich die Sensoren zu nullen. Diese Methoden unterscheiden sich untereinander lediglich durch die zugewiesenen Werte. Die programmtechnische Behandlung ist bei allen drei Methoden gleich. Aus diesem Grund wird auf die Darstellung der weiteren Methoden verzichtet.

Für die Radlast besteht die Besonderheit darin, dass die Gewichtskraft des Rades und der Schwinge im Zustand Rad oben einen negativen Wert ausgeben. Sind das Rad und die Hydraulikzylinder im Ruhezustand auf dem Boden ergibt sich im Kraftsensor ein Wert, der null ergibt. Es wirken aber bereits die Gewichtskräfte des Rades und der Schwinge. Aus diesem Grund muss der Ausgangswert als absoluter Betrag dem Messwert des Sensors hinzugerechnet werden.

6.2.12 Das Formular Reifen – das Bedienterminal in Software

Für den manuellen Betrieb wurde eine Oberfläche entwickelt, die das reale Bedienfeld darstellt und funktional identisch ist. Auf diese Weise ist es einfach möglich die Funktionalität der Steuerung zu überprüfen und verschiedene Schaltbefehle auszulösen.

Abbildung 44: Formular Reifen - Bedienterminal

Bei der Umsetzung der Druckknöpfe ist das Problem aufgetreten, dass die in der Entwicklungsumgebung vorgesehenen Objekte, vom Typ Button, Images als Bilder zwar unterstützen aber die Form dennoch quadratisch bleibt, was nicht besonders professionell aussieht, wie die folgende Abbildung zeigt:

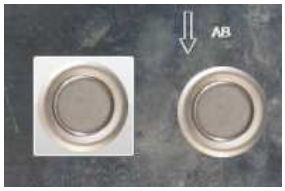


Abbildung 45: Button vs. Label

Im Internet findet sich eine Lösung Namens RoundButton [14]. Obwohl die Implementierung relativ lange zurück liegt, lässt sich das Modul wie beschrieben in eigene Projekte einbinden. Nachteilig wirkt sich jedoch aus, dass Teilprogramme geschützt sind und gerade dieser Teilbereich die Anwendung mehrfach zum Absturz gebracht hat. Aus diesem Grund musste eine bessere Lösung gefunden werden. Eher zufällig, durch die Implementierung einer LED im aus- bzw. eingeschalteten Zustand ist aufgefallen, dass der transparente Bereich, wie vorgesehen, auch zur Laufzeit transparent dargestellt wird. Eingesetzt wurde dafür das Label-Objekt. Das Label-Objekt verfügt, genau wie auch ein Button-Objekt, über Ereignisse vom Typ Click, MouseDown und

MouseUp. In diesem Zusammenhang wurden die Button-Objekte durch Label-Objekte ersetzt. Zusätzlich zur auszulösenden Aktion wurde ein Wechsel des Bildes vorgenommen, welches den gedrückten Zustand darstellt.

Als Image wurden unten dargestellte Bilder im png-Format mit transparentem Hintergrund angefertigt und verwendet. Für die Entwicklung der Bilder wurde die Freeware Paint.NET [15] eingesetzt.

Abbildung 46: Label mit transparentem Bild als Button

Die Software „hinter dem Formular“ enthält folgende Grundbestandteile:

- Einbindung der IOCP.Common-Klasse
- Instanziierung auf IOCPClient und die für den Prüfstand entwickelte Klasse Steuerung
- Initialisierung des Formulars und der Steuerung
- Ausführen des Hintergrundprozesses zur Überwachung des IOCP-Servers
- Festlegung der Ausgänge an deren Statusänderung man interessiert ist
- Verarbeitung der Ereignisse in der Methode OnVariableChanged und dort die Änderung der Bilder der LEDs oder Taster durchführt
- Methoden für die Reaktionen auf Tastendruck

```
using IOCP.Common;
:
:
public partial class Reifen : Form, IVariableChangeListener
{
 private IOCPClient _client = new IOCPClient();
 private Steuerung _mod1 = new Steuerung();

 public Reifen()
 {
     InitializeComponent();

     // Connect to the IOCP server
     _client.Connect(ConfigurationManager.AppSettings["IOCPServerAddress"], 8092);

     _client.AddVariableChangeListener(this);
     _client.AddVariableChangeListener(_mod1);

     // Runs the event dispatcher asynchronously in the background
     _client.RunEventDispatcherAsync();
```

```
    if (CZustand.EinAus)
        btnEinAus.Image =
                ((System.Drawing.Image)(Properties.Resources.Taste_Ein_pnet));
}

/// <summary>
/// This method is called, when one of the IOCP-Variables,
/// the client is interested in, is changed.
/// </summary>
/// <param name="Variable"></param>
/// <param name="State"></param>
public void OnVariableChanged(int Variable, int State)
{
 switch (Variable)
 {
   case (int)Outputs.LEDALARM:
   {
   if (State == 1)
    {
      lblLEDAlarm.Image = ((System.Drawing.Image)(Properties.Resources.LED_ein));
    }
    else
    {
      lblLEDAlarm.Image = ((System.Drawing.Image)(Properties.Resources.LED));
     }
     break;
   }
   case (int)Outputs.LEDEIN:
    {
       if (State == 1)
       {
          btnEinAus.Image =
                ((System.Drawing.Image)(Properties.Resources.Taste_Ein_pnet));
       }
       else
       {
         btnEinAus.Image =
                 ((System.Drawing.Image)(Properties.Resources.Taste_Aus_pnet));
       }
       break;
     }
    default: break;
  }
 }

public void Initialize(ISendData DataSender)
{
}

/// <summary>
/// Returns all the IOCP-Variables the client is interested in.
/// </summary>
public List<int> UsedVariables
{
get
 {
 List<int> result = new List<int>();
           result.Add((int)Outputs.LEDALARM);
           result.Add((int)Outputs.LEDEIN);
           return result;
 }
}
```

```
private void lblNotAus_MouseDown(object sender, MouseEventArgs e)
{
  CZustand.NotAus = !CZustand.NotAus;

  if  (CZustand.NotAus)
  {
    lblNotAus.Image = ((System.Drawing.Image)(Properties.Resources.NotAus_Ein));
    blinken();
    _client.SendData((int)Inputs.NotAus, 0);
  }
  else
  {
    lblNotAus.Image = ((System.Drawing.Image)(Properties.Resources.NotAus));
    blinken();
    _client.SendData((int)Inputs.NotAus, 1);
  }
}

private void btnEinAus_MouseDown(object sender, MouseEventArgs e)
{
  _client.SendData((int)Inputs.EinAus, 1);
}

private void btnEinAus_MouseUp(object sender, MouseEventArgs e)
{
  _client.SendData((int)Inputs.EinAus, 0);
}

private void btnInit_MouseDown(object sender, MouseEventArgs e)
{
  if (CZustand.EinAus)
  {
    btnInit.Image =
                 ((System.Drawing.Image)(Properties.Resources.Taste_down_pnet));
    _client.SendData((int)Inputs.Init, 1);
  }
}

private void btnInit_MouseUp(object sender, MouseEventArgs e)
{
  if (CZustand.EinAus)
  {
    btnInit.Image = ((System.Drawing.Image)(Properties.Resources.Taste_pnet));
    _client.SendData((int)Inputs.Init, 0);
  }
}
```

Die Funktionalität der Tasten kann in drei Gruppen gegliedert werden. Für den Ein-/Aus-Taster „btnEinAus“ wird lediglich beim Drücken eine 1 an den IOCP-Server gesendet, beim Loslassen eine 0. Der Wechsel des Status von Ein auf Aus bzw. umgekehrt erfolgt in der Steuerung. Diese Änderung teilt der IOCP-Server mit, worauf die Methode OnVariableChanged im Formular ausgeführt wird und im konkreten Beispiel je nach Status den Ein-/Aus-Schalter mit leuchtender LED oder im Normalzustand anzeigt.

Beim Druck auf den Taster Initialisierung, für Variante 2, wird zusätzlich zum Senden der 1 beim drücken und 0 beim Loslassen das Bild des Tasters geändert, so dass der optische Eindruck der Betätigung des Tasters entsteht. Beim Loslassen wird das normale Bild angezeigt. In der Steuerung wird die zugeordnete Init-Methode ausgeführt.

Die dritte Variante kommt beim Not-Aus-Schalter zum Einsatz. Hier wird das Bild entsprechend des Status gewechselt und gleichzeitig ein Timer gestartet, der die Alarm-LED zum Blinken bringt, indem das Bild der LED entsprechend wechselt. Die folgenden 2 Methoden zeigen den Ablauf.

```
private void blinken()
{
    if (CZustand.NotAus == true)
        timer1.Enabled = true;
    else
        timer1.Enabled = false;
}

private void timer1_Tick(object sender, EventArgs e)
{
    if (CZustand.ALARM == true)
    {
        lblLEDAlarm.Image = Properties.Resources.LED_ein;
        _client.SendData((int)Outputs.LEDALARM, 1);
    }
    else
    {
        lblLEDAlarm.Image = Properties.Resources.LED;
        _client.SendData((int)Outputs.LEDALARM, 0);
    }
    CZustand.ALARM = !CZustand.ALARM;
}
```

Im Formular selbst findet keine Datenspeicherung oder Datenänderung statt. Alle Aktivitäten werden über Nachrichten mit dem IOCP-Server ausgetauscht. Die Aktionen werden an den Server gesendet, dieser informiert das Formular über Änderungen an den Ausgängen, an denen das Formular interessiert ist und reagiert entsprechend.

6.2.13 Das Hauptformular – Reifenprüfstand

Über das Hauptformular (Abbildung 47: Hauptformular - Reifenprüfstand) können alle Funktionen bezüglich des Reifenprüfstandes aufgerufen werden.

Im Mittleren Bereich befinden sich die Bedienschaltflächen für den Aufruf der einzelnen Funktionen des Reifenprüfstandes. Im Unterschied zum Formular Reifen (siehe Abbildung 44) wird jede Aktion so lange ausgeführt, bis diese mit „Stop“ oder durch das Erreichen des jeweils gewählten Endzustandes eingenommen wurde.

Im rechten Bereich des Formulars können die Positionen des Messbalkens und des Tisches gewählt und eingenommen werden. Die aktuellen Positionen werden unterhalb der Schaltflächen „Position einnehmen“ angezeigt

Im linken Bereich des Formulars wird die Betriebsart ausgewählt, darunter befinden sich Eingabemöglichkeiten für gewünschte Sollparameter. Für die Steuerung sind insbesondere die Parameter Auflastkraft und die Messstrecke von Bedeutung. Diese Parameter beeinflussen das Verhalten des Prüfstandes direkt.

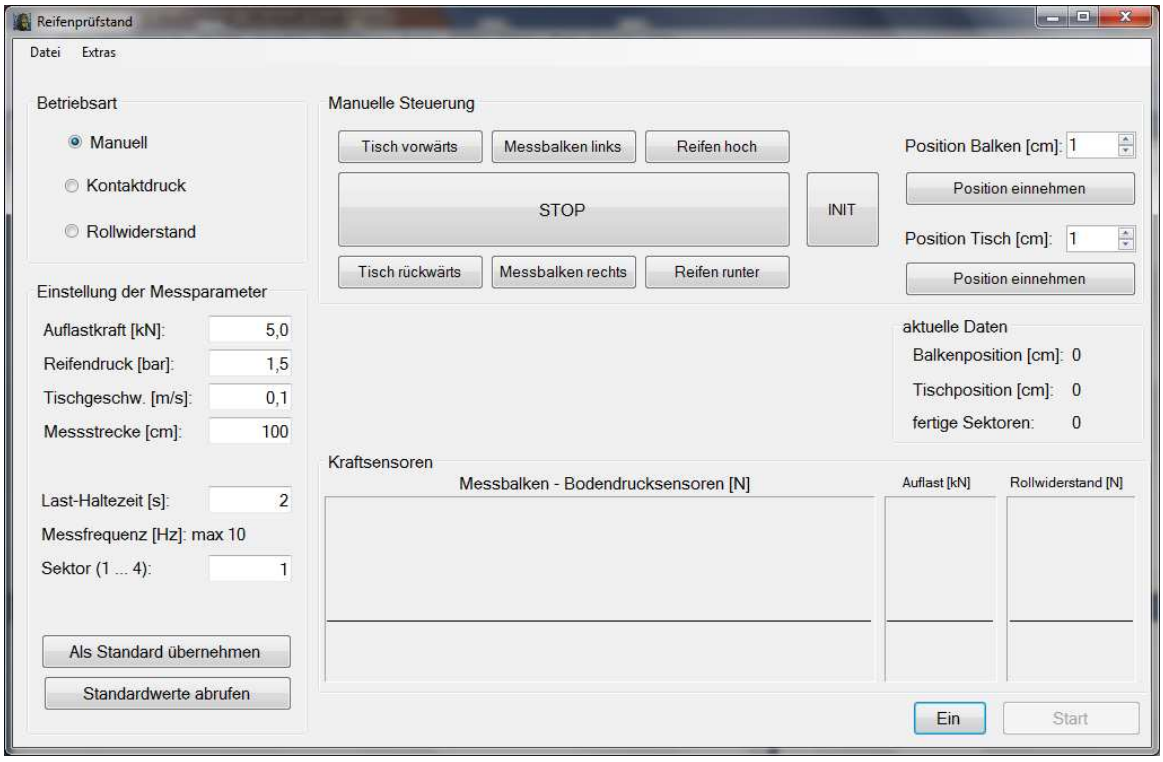


Abbildung 47: Hauptformular - Reifenprüfstand

Die Schaltfläche „Als Standard übernehmen“ erlaubt die Speicherung der aktuell eingetragenen Werte. Beim Aufruf des Formulars werden diese Werte wiederhergestellt. Die Schaltfläche „Standardwerte abrufen“ erlaubt das Rücksetzen auf Standardwerte, die in der Programmkonfiguration abgelegt sind.

Im Bereich „Kraftsensoren“ werden die aktuell ermittelten Werte der Minikraftsensoren im Messbalken, die aktuelle Auflast sowie der Rollwiderstand visualisiert. Es werden sowohl Farbsäulen entsprechend der Größe der Kräfte als auch die absoluten Angaben in N bzw. kN angezeigt.

Mit dem Taster „Ein“ wird die Kommunikation mit dem Steuercontroller (USB) und dem DataScan-Modul (serielle Schnittstelle) hergestellt und der Status auf Eingeschaltet gesetzt. Mit dem Einschalten werden die Tasten des Bereichs „Manuelle Steuerung“ aktiviert. Die Anzeige der Daten erfolgt ab diesem Moment im Bereich der Kraftsensoren sowohl visuell in einer Farbsäulenanzeige als auch in absoluten Werten oberhalb der Farbsäulen.

Die Einstellung der Kommunikationsparameter für die serielle Schnittstelle wird über den Menüpunkt Extras – Einstellungen… aufgerufen. Dort werden alle, zum Zeitpunkt des Aufrufs, im System vorhandenen seriellen Schnittstellen aufgelistet und in einer DropDown-Liste zur Auswahl gestellt. Die ausgewählte Schnittstelle wird mit deren Bestätigung übernommen. Nur bei einer Änderung des Anschlusses muss diese Konfiguration angepasst werden.

Über das Menü Extras können die Sensoren einzeln oder gemeinsam genullt werden. Dies ist notwendig, um die bereits angezeigten Werte bei der Ermittlung von Messdaten zu berücksichtigen.

Ebenfalls über das Menü Extras kann die Kalibrierung des DataScan-Modules ausgelöst werden. Dies ist Normalerweise nicht notwendig. Nur für den Fall, dass DataScan in den Auslieferungszustand gesetzt wurde muss diese Kalibrierung durchgeführt werden. Genau für diesen Zweck wurde diese Methode entwickelt und in die Klasse DataScan (siehe unter 6.2.8 Klasse DataScan, Event – Messwerte) integriert. Aus den genannten Gründen ist für die Ausführung eine Sicherheitsabfrage zu bestätigen, ehe dieser Vorgang tatsächlich ausgeführt wird. Die Minikraftsensoren im Messbalken und der Scherkraftsensor müssen bei diesem Vorgang im unbelasteten Zu-

stand sein, da die Kalibrierung ansonsten zu einer fehlerhaften Nullstellung im DataScan-Modul führt.
Alle zur Anzeige gebrachten und einstellbaren Werte wurden gegenüber der Lösung durch [2], in der Darstellung, an die Messgrößen angepasst, die in der Praxis vorkommen.

Über das Datei-Menü kann die Anwendung nach dem Ausschalten mit dem „Ein“-Taster beendet werden.

Die prinzipielle Funktionalität des Formulars bezüglich der Steuerung ist identisch mit dem Formular Reifen. Aus diesem Grund ist es nicht notwendig diese Funktionen an dieser Stelle noch einmal zu erläutern. Der komplette Programmcode befindet sich auf der beigelegten DVD-ROM zusammen mit der VisualStudio 2010 Pro-Version. Eine Lizenz kann im Rahmen der MSDNAA kostenlos bezogen werden.

6.2.14 Festlegung von Events zur Bekanntmachung der Positionsänderungen

Die Ermittlung der Positionen für den Messbalken und für den Tisch erfolgt in der Klasse Steuerung. Für die Anzeige und weitere Verarbeitung dieser Informationen, bspw. für die simple Anzeige dieser Informationen in einem Formular, muss dieses über die Änderung des Wertes informiert werden. Dies geschieht durch das Eventsystem. Ein Event wird in der folgenden Reihenfolge festgelegt und genutzt.

Als erstes wird ein Delegat deklariert, dessen Signatur den Typ des Ereignisses repräsentiert.

```
public delegate void TischPosChangedHandler(object sender,
                                             TischPosChangedEventArgs e);
```

Anschließend wird das Ereignis TischPosChanged definiert.

```
public event TischPosChangedHandler TischPosChanged;
```

Es wird die Methode definiert, in welcher das Ereignis ausgelöst wird.

```
protected void OnTischPosChanged()
{
  if (this.TischPosChanged != null)
     this.TischPosChanged(this,
```

```
                        new TischPosChangedEventArgs(CPosition.PosTischIst));
}
```

An der Stelle, an der dieser Event gefeuert werden soll, wird das Ereignis aufgerufen.

```
this.OnTischPosChanged();
```

Das definierte Ereignis wird im Formular Reifenprüfstand registriert. Ist kein Eventhandler registriert entfällt das Ereignis einfach.

```
_mod1.TischPosChanged +=new
                  Steuerung.TischPosChangedHandler(mySteuerung_TischPosChanged);
```

Im folgenden Eventhandler wird mit der anonymen Methode, per Invoke der Wert der Tischposition an das Label lblTischPos übergeben. Eine normale Zuweisung ist nicht möglich, da sich die Werte in unterschiedlichen Threads befinden.

```
private void mySteuerung_TischPosChanged(object sender,
                                                    TischPosChangedEventArgs e)
{
    lblTischPos.Invoke(new EventHandler(delegate
    {
        lblTischPos.Text = CPosition.PosTischIst.ToString();
    }
        ));
}
```

6.2.15 Visualisierung der Messwerte in Balken – ProgressBar vs. Panel

Für die Visualisierung der Messwerte stellt VisualStudio überraschenderweise kein Control zur Verfügung. Einzig die ProgressBar [16] zeigt ein annehmbares Verhalten. Es kann eine durchgehende Leiste, die von links nach rechts ausgefüllt wird anzeigen. Wenn die Minimum- und Maximum-Werte sowie der aktuelle Wert übergeben werden kann die Größe des Balkens gesteuert werden. Eine Eigenschaft, die eine vertikale Ausrichtung zulässt, ist nicht vorhanden. In vorhergehenden VisualStudio-Versionen war diese Möglichkeit noch vorhanden. Unter [16], http://social.msdn.microsoft.com/forums/en-US/winforms/thread/60b2493d-c8ff-495d-b845-d114fe456f54/ wird nachfolgende Möglichkeit bekannt gemacht:

```
using System;
using System.Windows.Forms;

public class VerticalProgressBar : ProgressBar {
  protected override CreateParams CreateParams {
    get {
      CreateParams cp = base.CreateParams;
      cp.Style |= 0x04;
      return cp;
    }
  }
}
```

Die Style-Eigenschaft des ProgressBar-Controls wird überschrieben. Durch die Oder-Verknüpfung mit dem hexadezimalen Wert 0x04 wird eine 90-Grad Drehung vereinbart. Dieses neue Control steht anschließend in der Entwicklungsumgebung als VerticalProgressBar zur Verfügung und kann wie alle anderen Controls benutzt werden.

Die Farbe des angezeigten Balkens sollte auf blau gesetzt werden. Diese Änderung kann jedoch nicht angezeigt werden, hatte aber keine Auswirkung auf den grün angezeigten Balken. Weitere Recherchen der MSDN [16] haben aufgezeigt, dass die Eigenschaft EnableVisualStyles deaktiviert werden muss, wenn die Farbeinstellungen übernommen werden sollen. Folgt man dieser Empfehlung ändert sich das Gesamtbild der Anwendung in eine Darstellung, wie diese aus Zeiten von Windows95 bzw. WindowsNT bekannt sind.

Ein weiterer Nachteil des ProgressBar-Controls besteht darin, dass nur positive Werte visuell angezeigt werden können. Bei der Nutzung müsste dies durch eine Verschiebung negativer Werte in den positiven Bereich berücksichtigt werden.

Unter Berücksichtigung aller Nachteile ist der Autor zu einer anderen Lösung gekommen. Es werden ganz einfache Panel verwendet, bei denen die gewünschte Farbeigenschaft und in Abhängigkeit der Messgröße die Höheneigenschaft manipuliert werden. Der Koordinatenursprung beginnt in der linken oberen Ecke der GroupBox mit der Bezeichnung Kraftsensoren. Die X-Achse verläuft von links nach rechts, die y-Achse von oben nach unten. Für die Nulllinie wurde die y-Position 150 festgelegt. An dieser Linie wurden die gewünschten Panels visuell platziert. Wird die Höhe des Panels vergrößert, „wächst“ der Balken von der festgelegten y-Position nach unten. Bei positiven Werten müssen zwei Parameter geändert werden. Von der y-Basisposition ist die Höhe abzuziehen und es ist die Höheneigenschaft zu setzten. Für die Darstellung der Balken stehen für positive Werte 95 Pixel zur Verfügung. Ausgehend vom Maximalwert von etwa 200 N, für die Minikraftsensoren im Messbalken, wird der Skalierungsfaktor von 2 ermittelt und als fy vereinbart.

Der folgende Programmausschnitt zeigt die Umsetzung. Der Variablen k1 wird die Balkenhöhe durch Anwendung des Skalierungsfaktors auf den Messwert zugewiesen. Das Panel mit der Bezeichnung panKraft1 betrifft den ersten Messbalken. Der Height-Eigenschaft wird der absolute Betrag übergeben. In Abhängigkeit des Vorzeichens wird die Position, wie oben beschrieben, ebenfalls korrigiert.

```
private int fy = 2; // Faktor für die Balken max. 200 N / 95 Pixel = 2
private int fAufl = 2; // Faktor Auflast max.      150 kN / 95 Pixel = 2

    int k1 = (int)CZustand.Kraft1 / fy;
    int auflast = (int)CZustand.Auflast / fAufl;
    int rollwid = (int)CZustand.Scherkraft / fRollWid;

    panKraft1.Height = System.Math.Abs(k1);
    if (k1 > 0)
        panKraft1.Location = new Point(16, 150 - k1);
    else
        panKraft1.Location = new Point(16, 150);

    panAuflast.Height = System.Math.Abs(auflast);
    if (auflast > 0)
        panAuflast.Location = new Point(512, 150 - auflast);
    else
        panAuflast.Location = new Point(512, 150);
```

6.2.16 Automatischer Modus – Kontaktdruck

Eine sehr wichtige Anforderung der Anwendungssoftware besteht in der Durchführung automatischer Prüfvorgänge. Das betrifft den Kontaktdruck sowie den Rollwiderstand. Nachfolgende Tabelle zeigt das Struktogramm für den Kontaktdruck.

Tisch frei? ja / nein
_AutoMod setzen
messstrecke = Convert.ToInt16(txtMessstrecke.Text)
startposTisch = CZustand.TischPosCenter – messstrecke / 2
CPosition.PosTischSoll = startposTisch
endposTisch = startposTisch + messstrecke
startposMB = 1
endposMB = CZustand.MBPosMax
SchwingeAuf
solange nicht SCHWINGEend
MessBalkenLinks
solange nicht MESSBAKENendLi
TischZur
solange nicht TISCHendVorn
für i = startposMB bis endposMB
MBPosAutoMod
solange PosMessbalkenIst kleiner als i
für j = startposTisch bis endposTisch
TischVor
so lange PosTischIst kleiner als j
SchwingeAb
so lange CZustand.AuflastSoll größer CZustand.AuflastIst
KontaktdruckSpeichern
SchwingeAuf
so lange nicht SCHWINGEend
TischZur
so lange nicht CPosition.PosTischIst größer als startposTisch
_AutoMod zurücksetzen

Tabelle 9: Struktogramm Kontaktdruck

Der folgende Programmausschnitt zeigt den Ablauf des Programms in den o. g. Schritten. Die einzelnen Abschnitte wurden im Code in der Form kommentiert, dass sich eine weitere Be-

schreibung erübrigt. Am Ausgang jedes Schrittes werden die Abbruchbedingungen geprüft. Der nächste Schritt wird also erst eingeleitet, wenn der vorherige abgeschlossen wurde. Die Einführung eines Wartezyklus – _datascan.delay(100) – hat sich notwendig gemacht, da sonst beim Durchlaufen der Schleife die gesamte Prozessorzeit zugewiesen wird. Eine Reaktion auf die eintretenden Events der Endschalter bzw. der Positionsänderungen ist dann nicht mehr möglich und das Programm ist nicht mehr bedienbar.

Der Programmteil wird in Abhängigkeit der gewählten Checkbox nach betätigen der „Start"-Taste ausgeführt.

```
else if (radioKontaktdruck.Checked)
{
    DialogResult result =
    MessageBox.Show("Kontaktdruck messen - Tisch frei?!", "Kontaktdruck",
    MessageBoxButtons.OKCancel,
    MessageBoxIcon.Question,
    MessageBoxDefaultButton.Button2);
    if (result == DialogResult.OK)
    {
        int messstrecke = Convert.ToInt16(txtMessstrecke.Text);
        int startposTisch = CZustand.TischPosCenter - messstrecke / 2;
        CPosition.PosTischSoll = startposTisch;
        int endposTisch = startposTisch + messstrecke;
        int startposMB = 1;
        int endposMB = CZustand.MBPosMax;

        _AutoMod = true;

        // 1. Reifen hoch
        do {
             _client.SendData((int)Inputs.SchwingeAuf, 1);
             _datascan.delay(100);
           } while (!CZustand.SCHWINGEend);
        // bis Schwinge am Endschalter/oben

        // 2. Messbalken an linken Anschlag fahren - Kalibrierung der Position
        do {
             _client.SendData((int)Inputs.MessBalkenLinks, 1);
             _datascan.delay(100);
           } while (!CZustand.MESSBAKENendLi);
        // bis Messbalken am Endschalter links

        // 3. Tisch an vorderen Endschalter fahren - Kalibrierung der Position
        do {
             _client.SendData((int)Inputs.TischZur, 1);
             _datascan.delay(100);
            } while (!CZustand.TISCHendVorn);
        // bis Endschalter TischVorn erreicht ist

        // Durchlaufen der Schleifen entsprechend der gewählten Daten
        // äußere Schleife - Messbalkenpositionen
        // innere Schleife - Tischpositionen
```

```
for (int i = startposMB; i < endposMB; i++)
{
    // 4. MB erste/nächste Position
    CPosition.PosMessbalkenSoll = i;
    // Eventhandler die gewünschte Position bereitstellen, dann STOPP
    CZustand.MBein = true;

    do
    {
        _client.SendData((int)Inputs.MBPosAutoMod, 1);
        if (statMB)
            _client.SendData((int)Inputs.MBPosAutoMod, 1);
        else
            _client.SendData((int)Inputs.MBPosAutoMod, 0);
        statMB = !statMB;
        _datascan.delay(100);
    } while (CPosition.PosMessbalkenIst < i);
    // bis nächste MBPos erreicht ist
    CZustand.MBein = false;

    for (int j = startposTisch; j < endposTisch; j++)
    {
        // 5. Tisch erste/nächste Position
        CPosition.PosTischSoll = j;
        // Eventhandler die gewünschte Position bereitstellen, dann STOPP

        CZustand.Tischein = true;

        do
        {
          _client.SendData((int)Inputs.TischVor, 1);
          _datascan.delay(25);
        } while (CPosition.PosTischIst < j);
        // bis nächste TischPos erreicht ist
        _client.SendData((int)Inputs.TischVor, 0);
        CZustand.Tischein = false;

        // 6. Reifen ab bis Sollwert erreicht ist
        CZustand.SchwingeEin = true;
        do
        {
            _client.SendData((int)Inputs.SchwingeAb, 1);
            _datascan.delay(100);
        } while (CZustand.AuflastSoll > CZustand.AuflastIst);
        // bis Auflast erstmalig erreicht ist

        // Messwerte erfassen und speichern
        KontaktdruckSpeichern();

        CZustand.SchwingeEin = false;
        _client.SendData((int)Inputs.SchwingeAb, 0);
        _datascan.delay(250);

        // 7. Reifen hoch
        do {
             _client.SendData((int)Inputs.SchwingeAuf, 1);
             _datascan.delay(100);
           } while (!CZustand.SCHWINGEend);
        // bis Schwinge am Endschalter/oben
        _client.SendData((int)Inputs.SchwingeAuf, 0);
        // Schleifenende für Tischpositionen
        }
```

```
            // 8. Tisch zurück zur startposTisch
            CPosition.PosTischSoll = startposTisch;
            // Eventhandler die gewünschte Position bereitstellen, dann STOPP
            CZustand.Tischein = true;

            do {
                 _client.SendData((int)Inputs.TischZur, 1);
                 _datascan.delay(100);
               } while (CPosition.PosTischIst >= startposTisch);
            // bis StartPosTisch erreicht ist
            CZustand.Tischein = false;
            // nächste Balkenposition
        }
    }
    _AutoMod = false;
}
```

6.2.17 Automatischer Modus – Rollwiderstand

Die folgende Abbildung zeigt das Struktogramm für den Rollwiderstand.

Tisch frei? ja	nein
_AutoMod setzen	
startposTisch = 1	
CPosition.PosTischSoll = startposTisch	
endposTisch = CZustand.TischPosMax	
SchwingeAuf	
solange nicht SCHWINGEend	
TischZur	
solange nicht TISCHendVorn	
SchwingeAb	
Tischvor	
RollwiderstandSpeichern bei jedem TischPosChanged-Event	
so lange Tischpos kleiner endposTisch ist	
so lange CZustand.AuflastSoll größer gleich CZustand.AuflastIst	
SchwingeAuf	
solange nicht SCHWINGEend	
TischZur	
solange nicht TISCHendVorn	
_AutoMod zurücksetzen	

Tabelle 10: Struktogramm Rollwiderstand

6.3 Datenspeicherung/-auswertung

Die Datenspeicherung erfolgt, indem die Messwerte für jede Radbelastung erfasst und in ein Datenarray abgelegt werden. Jede Datenzeile enthält neben den gemessenen Werten für die Minikraftsensoren im Messbalken bzw. die Werte für den Rollwiderstand und die Auflast auch die Systemzeit sowie die Positionsangaben für den Tisch und den Messbalken. In den „Kopfinformationen“ sind Daten zum gemessenen Reifen, den Luftdruck und das Datum enthalten. diese Angaben können bei Bedarf auf einfache Weise erweitert werden. Nach erfolgreichem Abschluss eines Messvorganges können die Arraydaten in eine beliebige Datei auf einem externen Datenträger oder lokal abgelegt werden. Die abgespeicherten Daten liegen im csv-Format vor und können somit in beliebigen Programmen, wie bspw. in Excel, nachträglich bearbeitet, ausgewertet und visualisiert werden.

Eine wesentliche Erweiterung der Möglichkeiten in Bezug auf die Bearbeitung der Daten bietet sich durch die Anbindung an eine Datenbank. Im Rahmen dieser Diplomarbeit ist das jedoch nicht möglich. Wesentliche Teile der Datenanalyse, die auf wiederholbaren Mechanismen beruhen, könnten somit bereits fest in die Anwendung integriert werden. Gut vorstellbar sind Export- und Importfunktionen sowie ein Berichtswesen mit integrierten Ausgabemöglichkeiten.

Der Autor empfiehlt die Nutzung einer SQL Server Compact Edition-Datenbank für lokale Daten. Das DBMS-System wird durch Microsoft kostenlos zur Verfügung gestellt. Alle notwendigen Werkzeuge stehen im VisualStudio zur Verfügung. Damit ist eine nahtlose Integration in die bestehende Anwendung möglich. Sollte später der Bedarf nach Mehrbenutzerzugriffen auf die gespeicherten Daten entstehen, ist eine Portierung auf einen, ebenfalls kostenlos zur Verfügung stehenden, SQL-Server-Express problemlos möglich. Alle Programmteile die Datenbank betreffend können weiter verwendet werden. Es wird lediglich die Verbindung angepasst.

7. Inbetriebnahme und Funktionsnachweis

Die Inbetriebnahme wird in mehreren Teilabschnitten durchgeführt. Erst nach erfolgreichem Abschluss wird der nächste Schritt eingeleitet.

Für eine erste Diagnose des Gesamtsystems sowie für die Programmierung wurde durch den Autor ein Diagnoseboard (siehe Abbildung 36) entwickelt. Dort wurden alle Eingabetaster, ein Taster zur Simulation des Endschalters der Schwinge, Anschlussklemmen für die Lichtschranken der Positionsbestimmung sowie eine NotAus-Brücke bestückt. Die Ausgänge wurden als LEDs ausgeführt. Mit dieser Diagnosekarte können alle Funktionen noch vor Anschluss am echten System geprüft und zukünftig zur Fehlersuche und Erweiterung eingesetzt werden.

Als Nächstes wird der Steuercontroller mit den benötigten Komponenten bestückt. Dafür wurde die Grundplatte des Gehäuses neu angefertigt. Auf der Grundlage der DXF-Datei der studentischen Arbeit [2] wurden alle notwendigen Anpassungen vorgenommen. Dabei wurden alle Baugruppen in der Form angeordnet, dass bei geöffnetem Gehäusedeckel der Zugang an die Anschlüsse möglich ist. Die einzige Ausnahme bildet die USB-Expasion-Card, die ihren Platz unterhalb der MasterCard gefunden hat. An dieser Karte werden keine, die Geräteanschlüsse betreffenden Anschlüsse benötigt. Die Anschlüsse der Aktoren, wie den Hydraulikventilen, dem Motoranschluss für den Messbalken und der Sensoren, wie der Positionsinformationen und dem Endschalter der Schwinge, befinden sich schraubbare Systemsteckverbinder an der Gehäuserückseite. Die Stromversorgung wurde über einen Kaltgerätesteckverbinder realisiert und mit einem Netzschalter versehen.

Die benötigen Ein- und Ausgänge werden über eine selbst entwickelte Anschlusskarte und zwei 40polige Flachbandkabel mit der MasterCard verbunden. Es gibt keine direkt in das Gehäuse führenden Kabel und Signalleitungen, die direkt am Controller angeschlossen werden.

Nach der Fertigstellung werden die Anschlusskabel der Sensoren und Aktoren angefertigt, angepasst und an das System angeschlossen.

Im nächsten Schritt erfolgt die Überprüfung der angeschlossenen Komponenten. Die aktive Nutzung der Hydraulikfunktionen wird erst nach erfolgreicher Prüfung der Signalpegel durchgeführt.

Die Funktionsprüfung des Endschalters und der Lichtschranken der Positioniersysteme ist einfach. Für diese Aufgabe wird der SIOC-Server gestartet und die IOCP-Console aufgerufen. Im „Log On“-Modus werden alle Informationen der Ein- und Ausgänge in Echtzeit angezeigt und protokolliert.

Nach der Prüfung des NotAus-Schalters 1 = nicht betätigt und 0 für ausgelöst und der Taster, die beim Betätigen eine 1 liefern, können auch die Anzeige-LED’s geprüft werden. Für diesen Test können Werte per Send abgesetzt werden. Auch hier wieder 1 für eingeschaltet und 0 für ausgeschaltet.

Als nächstes werden die angeschlossenen externen Geräte und Anschlüsse geprüft. Durch Betätigen des Endschalters an der Schwinge ändert sich der Status, genau wie bei den Tastern.

Für die Prüfung der Lichtschranken muss sichergestellt werden, dass diese offen sind. In diesem Fall wird in der IOCP-Console eine 1 geliefert. Im Moment der Unterbrechung der Lichtschranke, muss der Status „sauber“ zu 0 wechseln. An dieser Stelle ist der Test aller passiven Elemente abgeschlossen und es können die Antriebe geprüft werden.

Für den Motor und die Hydraulikkomponenten werden jeweils zwei Signale benötigt. Ein Signal ist für die Richtung verantwortlich, das zweite schaltet die Antriebe ein. Wird der Messbalken eingeschaltet bewegt sich dieser nach rechts, da die Richtung standardmäßig 0 ist. Wurde vorher die Richtung auf 1 gesetzt wird die Antriebsrichtung gewechselt und der Messbalken bewegt sich beim Einschalten nach links.

Für die Hydraulik sind zwei unabhängige Bereiche vorgesehen, einer für den Tischantrieb, der zweite für das Anheben bzw. Senken des Rades. Soll der Tisch in Fahrtrichtung vorwärts bewegt werden, muss die Tischrichtung auf 1 gesetzt werden, sonst auf null. Für die Schwinge ist die Aufwärtsbewegung in der Richtung auf 1 zu setzten. Für die Hydrauliksteuerung ist wichtig, dass zwingend zuerst die Richtung gewählt und anschließend der Einschaltvorgang erfolgt. Wird die Richtung bei eingeschaltetem Magneten geändert kommt es zu Fehlfunkionen.

Während der Bewegung des Tisches bzw. des Messbalkens kann die Funktion der Lichtschrankensysteme in Verbindung mit den Lochprofilen getestet und erforderlichenfalls eine Justierung der Lichtschranken vorgenommen werden.

Nach dem Start der Anwendungssoftware können in gleicher Reihenfolge die Einzelfunktionen sowie die korrekten Anzeigen getestet werden. Insbesondere das Ansteuern der jeweiligen Endpositionen, die gleichzeitig eine Autokalibrierfunktion besitzen. Jedes Mal, wenn eine Endposition erreicht wird, werden die Positionszähler auf die Anfangs- bzw. Endwerte gesetzt. Während des normalen Betriebes ist dieser Vorgang nicht notwendig, da die aktuell eingenommenen Werte beim Beenden des Programmes gespeichert und beim Neustart wiederhergestellt werden. Es bietet sich an, eine Verifizierung der Positionen nach Richtungswechsel zu überprüfen.

Während des Softwaretests kann man die IOCP-Console zur Kontrolle der Schaltvorgänge mitlaufen lassen. Man erhält auf diese Weise einen guten Überblick, ob alle Aktionen, die Steuerung betreffend, erwartungsgemäß ablaufen.

Die letzte manuelle Überprüfung betrifft die Radbelastung. Dabei wird getestet, dass Beim Erreichen des Sollwertes die Hydraulik abschaltet und nach Unterschreitung des Sollwertes, eine automatische Nachbelastung erfolgt.

Nun können die automatischen Modi geprüft werden. Um sicher zu gehen, dass alle Funktionen und Rahmenbedingungen korrekt gewählt sind, bietet sich ein Test mit kurzer Messstrecke und geringer Radbelastung an. Bei einer Messstrecke von bspw. 2 cm werden 20 Messvorgänge in einer überschaubaren Zeit durchgeführt.

Zum Abschluss werden die gespeicherten Messdaten einer Dateiprüfung unterzogen und testweise einem weiteren Verarbeitungsprozess unterzogen.

Sind alle Funktionsnachweise erbracht, können Messdaten im Normalbetrieb aufgenommen werden.

8. Zusammenfassung und Ausblick

8.1 Beschreibung und Auswertung des Gesamtentwurfs

- Entwicklung und Austausch der Positioniersysteme auf Basis eines Lichtschrankensystems in Verbindung mit Lochleisten, ein System für die Ermittlung der Tisch- und Messbalkenpositionen
- Austausch des Mikrocontrollers durch ein HardwareBoard von OpenCockpits [3]
- das HardwareBoard abstrahiert die elektrischen Bauteile, wie Taster, LEDs, die Motortreiber- und die Hydraulikplatine komplett
- Entwicklung und Austausch der Trägerplatte der Bedieneinheit und Unterbringung der elektronischen Baugruppen, bei gleichzeitiger Verbesserung des Zugangs zu den Treiberplatinen
- Anschluss der Sensoren und Aktoren durch ein einheitliches Stecksystem an der Rückseite der Bedieneinheit, alle Anschlüsse erfolgen an der Rückseite der Bedieneinheit
- Übernahme und Auswertung der Messdaten vom DataScan direkt am PC über die serielle Schnittstelle
- Neuentwicklung der Software für die Steuerung, Bedienung und Datenerfassung des Prüfstandes auf Basis der Objektorientierten Programmiersprache C#, mit erheblicher Performancesteigerung, bei deutlich geringeren Anforderungen an den verwendeten PC
- modularer Aufbau der Software, der einen Austausch einzelner Klassen (Teilprogramme) unterstützt
- Wegfall von zusätzlich benötigten Lizenzen bspw. für MatLAb

8.2 Vorschläge für Erweiterungen und Anpassungen weiterer Prüfelemente

Durch den modularen Aufbau des Gesamtsystems, der elektronischen Baugruppen, als auch durch den Aufbau der Software, werden Erweiterungsmöglichkeiten auf einfache Weise ermöglicht.

Bei der Ermittlung des Rollwiderstandes, müssen für die Erfassung der Kenndaten über den gesamten Umfang des Rades, mehrere aufeinanderfolgende Messungen durchgeführt werden. Zwischen den Messvorgängen wird das Rad händig auf den nächsten Sektor eingestellt. Die Messdaten werden anschließend, manuell, den Reifensektoren zugeordnet.

Dieser Vorgang kann aus Sicht des Autors in der Form automatisiert werden, indem für die Achse bspw. ein elektrisches Bremssystem entwickelt wird. Damit kann die Voraussetzung für eine programmgesteuerte Funktion geschaffen werden. Die Treiberplatine für den Antriebsmotor verfügt über einen zzt. nicht genutzten, vollständig beschalteten Anschluss, der für den Schaltvorgang genutzt werden kann. Eine Einbindung über das HardwareBoard von OpenCockpits [9] kann durch Zuweisung von freien Ausgabeports, dessen nachträgliche Beschaltung, Festlegung von IOCP-Variablen und anschließender Integration in die Anwendungssoftware, eingebunden werden. In der Anwendungssoftware sind die notwendigen Steuerungssequenzen zu programmieren.

Weitere Erweiterungen bietet das DataScan-Modul. Das DataScan 7010 stellt insgesamt 24 D/A-Eingänge zur Verfügung, von denen 17 genutzt werden. Entsprechend dem Anschlussschema nach Abbildung 16, können weitere Sensoren für die Erfassung von Kennwerten eingebunden werden. Eine Kalibrierungssequenz kann entsprechend der Ausführungen unter 5.1 Sensoren durchgeführt werden. Damit ist es möglich, die elektrischen Signale direkt in gewünschte Messwerte umzuwandeln.

Für die Übernahme der dazugehörigen Messdaten im Anwendungsprogramm ist die Klasse DataScan anzupassen. Die Visualisierung dieser Daten kann anschließend in die Benutzeroberfläche übernommen werden. Alle notwendigen Methoden sind bereits beispielhaft vorhanden und müssen für die Anzeige lediglich um diese Daten ergänzt werden. Die erweiterte Lösung kann parallel entwickelt und getestet werden. Erst wenn Tests erfolgreich absolviert wurden wird die neue Anwendungslösung eingesetzt. Ein paralleler Betrieb verschiedener Softwarestände wird vollständig unterstützt.

Die Implementierung von Erweiterungen bspw. zur Erfassung von Querkräften oder von Drehmomenten zur Auswertung weiterer Kenndaten der Reifenverformung ist integrierbar.

Wie bereits im Referat erläutert wurde, weisen landwirtschaftliche Reifen teilweise erhebliche Deformationen auf. Kenndaten diesbezüglich werden zzt. nicht erfasst. Die Reifenhersteller geben lediglich Daten für den Reifeninnendruck, in Bezug auf die Belastbarkeit und Geschwindigkeit, vor. Konkrete Kenntnisse über die Art und Größe der Verformungen lassen Rückschlüsse auf die Reifenkonstruktion oder Fahrwerkskonstruktionen zu. Zur Erfassung dieser Daten ist es

notwendig die Reifen mit definierten Lasten zu prüfen. Dies ist mit dem Prüfstand möglich. Für die Erfassung der Verformung könnten optische Messverfahren eingesetzt werden, die den Verformungsverlauf in Echtzeit erfassen können. Eine Einbindung in die Software sowie eine Synchronisation der Messwerterfassung, mit Erreichen der gewünschten Belastung, sind denkbar.

Anlagen

Anlage 1 – Genehmigung zur Verwendung der IOCP-Wrapper-Klasse durch Klaus Aschenbrenner

Von: Klaus Aschenbrenner <klaus.aschenbrenner@csharp.at>
An: '"Bernd Müller"' <be-mueller@gmx.de>
Betreff: RE: RE: .NET steuert Hardware-Boards
Datum: 03.10.2011 21:44:14

Hallo Herr Müller,

Ja klar, ist überhaupt kein Problem, dass Sie meinen IOCP-Wrapper einsetzen :-)

Vielen Dank & liebe Grüße
Klaus Aschenbrenner

SQL Server Consultant, MCITP SQL Server 2008 Database Development & Administration
http://www.csharp.at | http://twitter.com/Aschenbrenner
Phone: +43 699 191 157 99

-----Original Message-----
From: "Bernd Müller" [mailto:be-mueller@gmx.de]
Sent: Montag, 03. Oktober 2011 21:14
To: klaus.aschenbrenner@csharp.at
Subject: Re: RE: .NET steuert Hardware-Boards

Sehr geehrter Herr Aschenbrenner,

vielen Dank für die Antwort, auch wenn ich schon gar nicht mehr damit gerechnet habe. OK, zwischenzeitlich läuft alles wunschgemäß und ich habe fast alles für meine Anwendung neu geschrieben. Es sind ein paar Kleinigkeiten im Hardwareemulator drin. Für mich war‘s schwierig mich da reinzupfitzen.

Ich möchte aber gern Ihren IOCP-Wrapper für dieses Projekt einsetzten und hätte dafür gern eine "Freigabe/Genehmigung" dafür, selbstverständlich mit der entsprechenden Quellangabe.

Sinn des Projektes besteht in der "Automatisierung" eines Prüfstandes an der TU Dresden im Rahmen meiner Diplomarbeit. Dieser Prüstand für Off-Road-Reifen, wie diese in der Landtechnik eingesetzt werden, ist einmalig. Es wird kein Produkt daraus. Für die Steuerung nehme ich, auf Grund des Artikels in der dotnetpro, ein Board von OpenCockpits.

Wäre schön, wenn ich nicht ganz so lange auf eine Antwort warten müsste.

Vielen Dank.
Mit freundlichen Grüßen
Bernd Müller

Literaturverzeichnis

[1] H. Döll, „Methode zur Bewertung von Landwirtschaftsreifen," 1999.

[2] M. Jaster, „Studienarbeit," 2010.

[3] OpenCockpits.com, „opencockpits.com," 2011. [Online].

[4] burster präzisionsmesstechnik gmbh & co kg, „http://burster.de," [Online].

[5] Datascan Technology, „Datascan Installation and User Guide," 1995.

[6] Tatham Simon http://www.chiark.greenend.org.uk/~sgtatham/putty/, „http://www.chiark.greenend.org.uk/~sgtatham/putty/," [Online].

[7] Datascan Technology, „Datascan7000-commands.pdf," 2002. [Online].

[8] K. Aschenbrenner, „Hardwareprogrammierung - .NET steuert Hardware," *dotnetpro,* p. 18 ff., 5 2010.

[9] OpenCockpits, „manual_expansion_master_eng.pdf," 2010.

[10] Conrad Electronic, „www.conrad.de," [Online].

[11] IceTux, 17 04 2007. [Online].

[12] „www.pepperl-fuchs.de," [Online].

[13] Visual C# 2010, W. Doberenz und T. Gewinnus, Grundlagen und Profiwissen, München: Carl Hanser Verlag, 2010.

[14] Saikat Sen, „CodeProject.com," 2003. [Online].

[15] paint.NET, „http://www.getpaint.net," 2011. [Online].

[16] Microsoft, „http://msdn.microsoft.com," 2011. [Online].

Erklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Dresden, 30.11.2011